TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Automating Telephony Testing for Integrated Access Devices (IADs)

Diploma Thesis

Study programme:	N2612 – Electrical Engineering and Informatics
Study brunch:	3906T001 – Mechatronics

Author:	**Filip Burda**
Supervising Professor:	Prof. Dr.-Ing. Dietmar Scharf
Supervisor:	Dipl.-Ing. (FH) Thomas Haak

Liberec 2015

Technical University of Liberec
Faculty of Mechatronics, Informatics and Interdisciplinary Studies
Academic year: 2014/2015

DIPLOMA THESIS ASSIGNMENT

(PROJECT, ART WORK, ART PERFORMANCE)

First name and surname:	**Bc. Filip Burda**
Study program:	**N2612 Electrical Engineering and Informatics**
Identification number:	**M12000266**
Specialization:	**Mechatronics**
Topic name:	**Automating Telephony Testing for Integrated Access Devices (IADs)**
Assigning department:	**Institute of Mechatronics and Computer Engineering**

Rules for elaboration:

The goal is to develop an automated telephony testing system for integrated access devices.
The following steps should be completed:

1. Choosing the right hardware (ISDN card) for testing.
2. Creating of a test setup.
3. Automation of call initiating.
4. Developing a program for automated stress tests.
5. Displaying test results.

Declaration

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo. Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL. Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem. Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Abstract

This Master Thesis is focused on telephony testing. At first, it describes fundamentals of telephony itself and especially ISDN and VoIP. The goal here is to develop an automated telephony testing system for devices produced by Sphairon GmbH (a ZyXEL Company). The thesis shows importance of testing in a development process. Also a procedure of choosing the right hardware for the task is presented. Then all the requirements for the chosen test setup (Asterisk, Linux Call Router, mISDN, ISDN card) are described as well as their configuration so they work together and are able to simulate incoming and outgoing calls for device under test. The next part of this document is about a development of the program for control of the test setup. The program automates the telephony testing process and gets information from Asterisk that is used for evaluating of the initiated calls. Last chapter shows output of the system which displays results that can be further inspected and evaluated because all the necessary files for that are archived. The automated telephony testing system is used to determine quality of tested devices.

Keywords: ISDN, VoIP, Telephony, Testing, Automated testing

Abstrakt

Tato Diplomová práce je zaměřena na testování telefonie. Nejdříve jsou zde popsány základy právě telefonie a to především ISDN a VoIP. Cílem této práce je vyvinout systém pro automatizované testování telefonie v zařízeních vyrobených společností Sphairon GmbH (a ZyXEL Company). Práce poukazuje na důležitost testování v procesu vývoje. Je zde také prezentován postup výběru správného hardwaru pro tento úkol. Dále jsou zde popsány všechny požadavky pro zvolenou testovací sestavu (Asterisk, Linux Call Router, mISDN, ISDN karta) stejně jako jejich konfigurace tak, aby pracovaly společně a bylo možné simulovat příchozí a odchozí volání pro testované zařízení. Další část tohoto dokumentu se zabývá vývojem programu pro ovládání této testovací sestavy. Tento program automatizuje proces testování telefonie a získává informace z Asterisku, které jsou použity pro vyhodnocování zahájených volání. Poslední kapitola ukazuje výstupy ze systému, kde jsou zobrazeny výsledky, které mohou být dále prozkoumány a vyhodnoceny, protože všechny nezbytné soubory z testu jsou archivovány. Automatický systém testování telefonie slouží pro stanovení kvality testovaných zařízení.

Klíčová slova: ISDN, VoIP, Telefonie, Testování, Automatizované testování

Contents

List of Figures

List of Codes

List of Abbreviations

ADPCM	Adaptive Differential Pulse-Code Modulation
AMR	Adaptive Multi-Rate compression
AOC	Advice of Charge
BRI	Basic Rate Interface
BSD	Berkeley Software Distribution
CDPN	Called Party Number
CELT	Constrained Energy Lapped Transform
CFx	Call Forwarding
CGPN	Calling Party Number
CLIP	Calling Line Identification Presentation
CLIR	Calling Line Identification Restriction
CW	Call Waiting
DSL	Digital Subscriber Line
DUT	Device Under Test
ET	Exchange Termination
GSM	Global System for Mobile communication
HFC	Hyper Fiber Chip
HTTP	Hypertext Transfer Protocol
IAD	Integrated Access Device
IAX	Inter-Asterisk eXchange
iLBC	Internet Low Bitrate Codec
IP	Internet Protocol
ISDN	Integrated Services Digital Network
LCR	Linux Call Router
LPC	Linear Predictive Coder

MGCP	Media Gateway Control Protocol
mISDN	Modular ISDN
NAT	Network Address Translation
NT	Network Termination
PBX	Private Branch Exchange
PMP	Point to Multipoint Protocol
POTS	Plain Old Telephone Service
PPP	Point to Point Protocol
PRI	Primary Rate Interface
PSTN	Public Switched Telephone Network
RTP	Real-time Transport Protocol
SCCP	Skinny Call Control Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMTP	Simple Mail Transport Protocol
SRTP	Secure Real-time Transport Protocol
SSH	Secure Shell
STUN	Session Traversal Utilities for NAT
TA	Terminal Adapter
TE	Terminal Equipment
TLS	Transport Layer Security
UNIStim	United Networks IP Stimulus
URI	Uniform Resource Identifier
VoIP	Voice over Internet Protocol
XMPP	Extensible Messaging and Presence Protocol
ZRTP	Zimmermann’s Real Time Transport Protocol

Introduction

This thesis is about developing an automated test for devices which are produced by Sphairon GmbH (a ZyXEL Company) and work as SIP gateways and IADs. Testing is an important part of every development process and in telephony it is not any different. All the new software releases have to pass a set of complex tests before they can go to a market. The outcome of this work is to be included in these tests to replace and extend a manual way of testing which was used before.

For testing of a software a regression test is used, that investigates if the version of the software introduced new faults and if the old errors reappeared. If there is a problem, the new software version needs to go back into development. If all the tests succeed, the software can be delivered to a customer.

Stress testing is a necessary part of regression tests. As the name implies, in this case, a device is stressed in every possible way to ensure that even in unusual circumstances everything works as it should. Stability of the device is determined by scale of intense tests.

The telephony testing is based on automatically created phone calls with different parameters that are sent from PC. The PC should act as any ISDN telephone. The goal here is to find a way to simulate that. At first a necessary software and hardware needs to be chosen. In this case it means an ISDN card with required properties and a software which can control it. Then the right configuration has to be made, so all the software and hardware works together. And finally a program has to be developed, which can control the whole setup, create automated calls based on given parameters, evaluate the success or fail of these calls and export results. Also all available information from the test and a device under the test have to be extracted so the results can be inspected and any fault can be fixed.

1. Fundamentals of Telephony

Telephony is a technology for electronic transmission of voice, fax and possibly other information at long distance. There are different types of telephony. First type is classic analog transmission. Another way is use of digitalized telephone network - ISDN (Integrated Services Digital Network). There is also a special part of digital telephony called VoIP (Voice over Internet Protocol) which uses transmission over Internet lines.

Besides the main data carried through the network during telephone contact, there are also data used for signalling. Signalling is exchange of information for maintaining the telephone call (setting up, controlling, terminating). In in-band signalling is for the signalling data used the same channel as for the telephone call. In contrast with out-of-band signalling which has its own separated channel. [1]

1.1 Analog Telephony

The analogue telephony works in analogue PSTN (Public Switched Telephone Network) which is also called POTS (Plain Old Telephone Service). These days most part of the network (if not whole) is digital and the only analog part which remains is subscriber connection. The reason is obvious. Analog telephony has to deal with all disadvantages of analog transmission like sensitivity to distortions.

Analog telephony does not necessarily need provider of PSTN. It can also work in private systems – local loops. In these kinds of systems the switching is managed by PBX (Private Branch Exchange) systems. It allows to make private telephone networks for instance in the range of some company. [2]

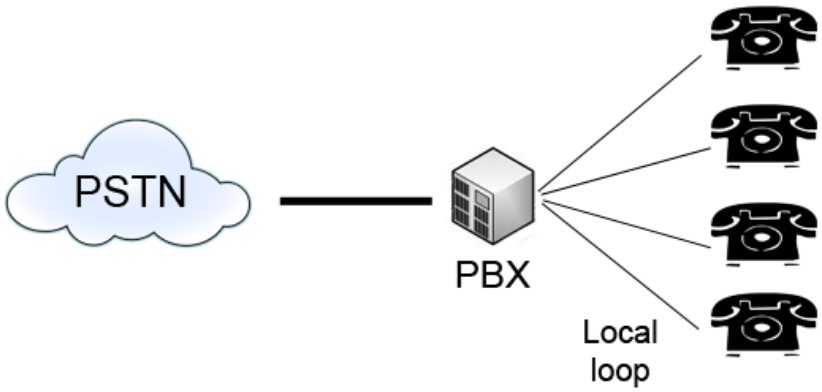

Fig. 1: Structure of telephone network with local loop

1.2 ISDN

Integrated Services Digital Network – ISDN is a digital telephony system. It can transmit voice, video and data through the network. Its advantages besides the typical benefits of digital transmission are higher speed and out-of-band signaling. [3] ISDN has various options and properties. It can work in two types of service levels which use various number of different channels.

1.2.1 ISDN Service Levels

The ISDN service levels differ by types and number of used ISDN channels. These properties also specify the rate of the service level.

The Bearer channel (B-channel) with 64 kbps carries the main information which are data, voice and video. For higher bandwidth the channels can be aggregated together.

The Delta channel (D-channel) can operate at 16 or 64 kbps. The bandwidth depends on used service level of ISDN. This channel handles signaling information needed to connect and disconnect calls and other services.

There is also a special high-speed H-channel for video transfer. There are four kinds of H-channels with rate 384 kbps, 1472 kbps, 1536 kbps and 1920 kbps. [3]

The Basic Rate Interface (BRI) has two independent B-channels for main data and one 16 kbps D-channel for signaling. This means the overall rate is 144 kbps (plus 48 kbps for maintenance and synchronization).

The Primary Rate Interface (PRI) service level differs in several parts of the world. In North America and Japan, it has 23 B-channels and one 64 kbps D-channel. In Europe and Australia PRI uses 30 B-channels and one 64 kbps D-channel. So PRI can operate at 1536 or 1984 kbps. [3]

1.2.2 ISDN Devices

Terminal Equipment (TE) is a communicating device that complies with the ISDN standards. It can be for instance digital telephone, ISDN data terminals or ISDN-equipped computer.

Terminal Adapter (TA) allows communicating devices that do not conform to ISDN standards to communicate over the ISDN.

Network Termination (NT1 and NT2) forms the physical and logical boundary between the customer premises and the carrier's network. NT1 performs logical and NT2 physical interface. Usually both functions are performed by one device – NT.

Exchange Termination (ET) makes the physical and logical boundary between the digital local loop and the carrier's switching office. [3]

1.2.3 ISDN Interfaces

- R interface – between a non-ISDN terminal device and a terminal adapter.
- S interface – between a terminal equipment and a network termination device
- T interface – between a network termination device 1 and 2
- U interface – between a network termination device and the carrier's local transmission loop. [3]

1.3 VoIP

VoIP stands for Voice over Internet Protocol and it is also addressed as IP telephony. It uses internet network for transmitting a digitalized voice so there is no "fixed" connection like in classical telephony. Instead, the voice is transmitted in packets over the internet protocol. This means that VoIP uses packet switching in contrast to circuit switching used in classical telephony. The advantages of VoIP are lower cost of calls and higher speed of data which allows to include more services. The disadvantage can be quality of a call. The quality is monitored using parameter of the call like latency, jitter and packet loss. The data are digitalized using codecs and function of VoIP is controlled by signalling protocols. The most common protocols are H.323, MGCP and especially SIP. [4]

SIP - Session Initiation Protocol is application-layer control protocol that is used for controlling real-time multimedia sessions like IP telephony. SIP for providing VoIP uses some other protocols (SDP, RTP). It is simpler alternative to H.323 and it is based on protocols like HTTP and SMTP. SIP uses HTTP request – response model and URI (Uniform Resource Identifier) similar to email address from SMTP.

A communication in SIP implements a three-way handshake. At first a caller sends an INVITE message and a callee returns OK to accept the call. Then the caller confirms the call by ACK message. In (Fig. 2) is shown example of SIP session with names of used methods and numbers of response codes. The session is between two users - Jane uses hardware SIP phone with SIP URI sip:jane@callfree.com and Mike has softphone in his PC with SIP URI sip:mike@myphone.cz. Where callfree.com and myphone.cz are their SIP service providers. The transaction starts with Invite message from Jane to Mike which at first sent to Jane's provider. The next step is sending the message from callfree.com to myphone.cz. This server knows location of Mike's softphone and sends the Invite there. Along the way both SIP servers send back Trying message with code starting with number one (100) which means that request was received and is being processed. The phone on Mike's side is ringing sends back Ringing message with code 180 again through both proxy servers. When the call is answered Mike's softphone sends an OK message with code 200. Messages which start with number two mean the action was successfully received, understood, and accepted. Jane's SIP phone answers with acknowledgement message but this time straight to Mike's softphone because the locations are already known. The media session itself starts after receiving acknowledgment. After hang-up the Bye message is sent and it is confirmed by 200 OK message. Besides message

codes starting with 1 and two there are other types of messages. Message code starting with 3 means that further action needs to be taken to complete the request. 4xx code signifies that request contains bad syntax or cannot be fulfilled at the server, 5xx indicates that server failed to fulfill an apparently valid request and codes starting with number 6 implies that request cannot be fulfilled at any server. [5]

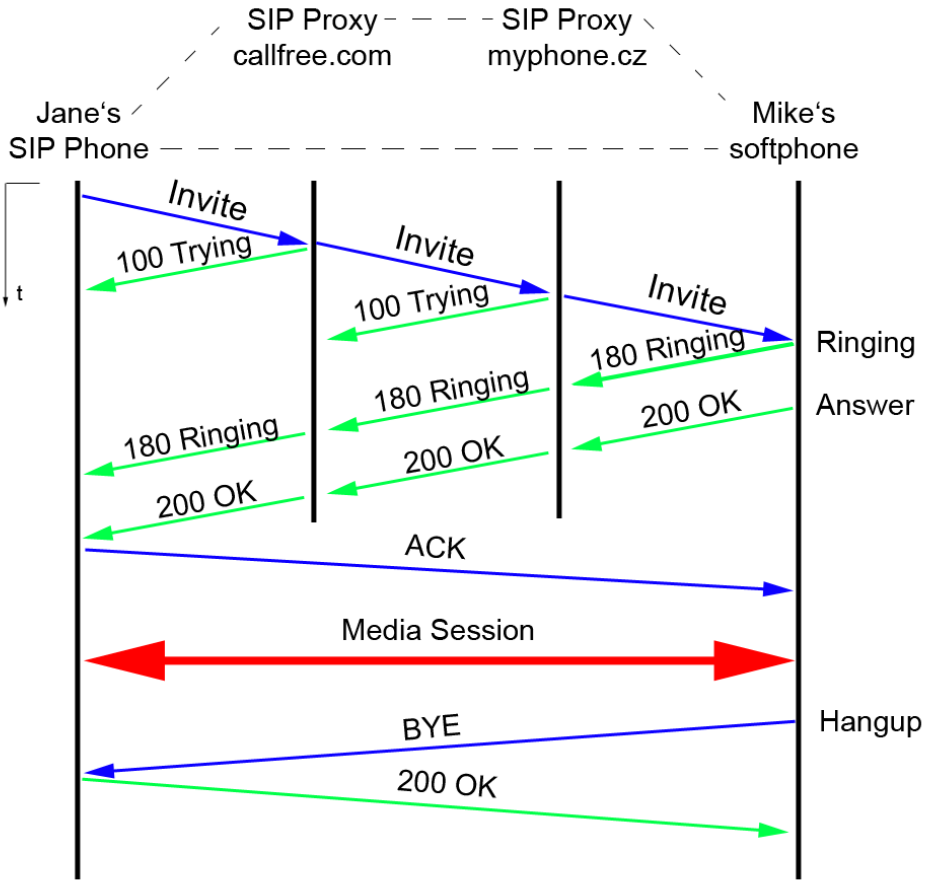

Fig. 2: SIP session example

1.4 Softswitches

Softswitch is a device which connects telephone calls in a telecommunications network. It is a central device in the network. A softswitch is typically used in IP telephony and it can provide a lot of functions and possibilities. Its advantage is in programmability which can of course differ from one product to another. Further on will be described basic properties of four tested softswitches – Asterisk [6], FreeSwitch [7], Yate [8] and Amooma Gemeinschaft [9]. There are also a commercial products – Adore Softswitch [10], Dialogic Softswitch [11], Technicolor Cirpack Softswitch [12] and many more.

First and maybe even most important thing when installing and using a new softswitch is its documentation. The documentation is usually on a wiki website and it contains notes for installation and basic and advanced configuration. Installation can be an easy task but it is not always so. The next step is setting up the system for basic calls. This can be done by configuration files which have a defined structure. The softswitch is in most cases controlled from a command line/terminal but it can be also expanded to some graphical user interface for more user friendly configuration and control. Very important property of every softswitch is its list of supported VoIP protocols. Most of them supports SIP, H.323 or IAX which is sufficient in most cases. There is also list of supported codecs which is obviously the longer the better. The next property can be type of signalling. In-band signalling uses the same channel as the call itself. Out-of-band signalling has its own channel. Besides these two types there are special ones like IAX2 or SIP-INFO. All of the tested systems support voice announcements and interactive voice response. These options are pretty much self-explanatory. Important feature these days is support of IPv6 since the whole internet world is moving that way. Some of the softswitches support protocol STUN (Session Traversal Utilities for NAT). It enables a device to find its public IP address. [13] The softswitches support various types of encryption like TLS or SRTP. None of the tested systems offers SIP-Trunking which is method where provider assigns range of numbers to a user and the user can divide them at will.

In (Tab. 1) is a comparison matrix for the tested softswitches where all of them were tested. They were evaluated by marks according experience of installation and use for basic calls from the point of view of a new user.

Comparison of softswitches: 1 – good (easy), 2 – medium, 3 – bad (difficult)				
	Asterisk	FreeSwitch	Yate	Amooma
Documentation	1	1	1	3
Installation	1	1	1	3
Set-up	1	2	1	3
Graphical user interface	Yes	Yes	Yes	Yes
Supported VoIP protocols	SIP, H.323, IAX,XMPP, Jingle MGCP, SCCP, UNIStim	SIP, H.323, IAX, XMPP, Jingle, SCCP, Skype	SIP, IAX, H.323, XMPP, Jingle, MGCP	SIP, ?[1)]
Voice announcements	Yes	Yes	Yes	Yes
Interactive voice response	Yes	Yes	Yes	Yes
Supported codecs	ADPCM, CELT, G.711, G.719, G.722, G.723, G.726, G.729a, GSM, iLBC, Linear, LPC-10, Speex, SILK	AMR, CELT, G.711, G.722, G.723, G.726, G.729AB, GSM, iLBC, LPC-10, Speex, SILK, DVI4, OPUS	AMR, GSM, iLBC, Speex	?[1)]
DTMF	Inband, Out -of-band, SIP-INFO, IAX2	Inband, Out-of-band, SIP-INFO, IAX2	Inband, Out-of-band, SIP-INFO	?[1)]
IPv6 support	Yes	Yes	Yes	No
STUN support	Yes	Yes	Yes	?[1)]
Operating systems	Linux, Mac OS, *BSD, Windows, Solaris	Linux, Mac OS, *BSD, Windows, Solaris	Linux, Mac OS, *BSD, Windows	Standalone system
Encryption	TLS, SRTP	TLS, SRTP, ZRTP	TLS	?[1)]
SIP trunk	No	No	No	No
[1)] Missing information due a poor documentation.				

Tab. 1: Softswitch comparison matrix

2. Why Testing?

Testing is very important part of every development process. Testing helps to find bugs in software as well as any issues caused by hardware. It can reveal problems introduced by new features in the developed product (regression testing) and also it gives a developer "user experience feeling". It means that the developer sees the product from a point of view of a user which helps him improve the product. Besides the regression testing, when new feature is added into a software it also has to be tested to find out if it has the expected outcome. When a customer has any requirements, all these have to be tested. In a development process, the essential thing is to perform the tests continuously so every new version is tested. This applies for special parts of the product as well as for the whole system with all its features. All these procedures lead to one key goal – to ensure the quality of the product is as good as possible. That's why every product have to be tested before it goes on the market. There are various types of testing:

- Black-box x Gray-box x White-box testing which differs according to the level of internal structures testing.
- Functional x Non-functional tests where the function of the system is tested or on the other hand the non-functional requirements of the system.
- Regression x Non-regression testing is testing whether the update or patch did not introduce new bugs vs. testing if the update or patch had desired effect
- Unit x Integration testing is testing of isolated parts of the system vs. testing the parts combined together
- Special tests e.g. Security, Stress, Endurance, Compatibility, Performance, Load, Recovery, Boundary, …

2.1 Testing in Sphairon

This thesis is focused on stress and endurance testing of the telephony system. A development process which includes among other things the regression testing is shown in (Fig. 3). Every update goes to regression testing system where it is tested whether it did not introduce new bugs. In Sphairon GmbH (a ZyXEL Company) a Jenkins system is used for the testing.

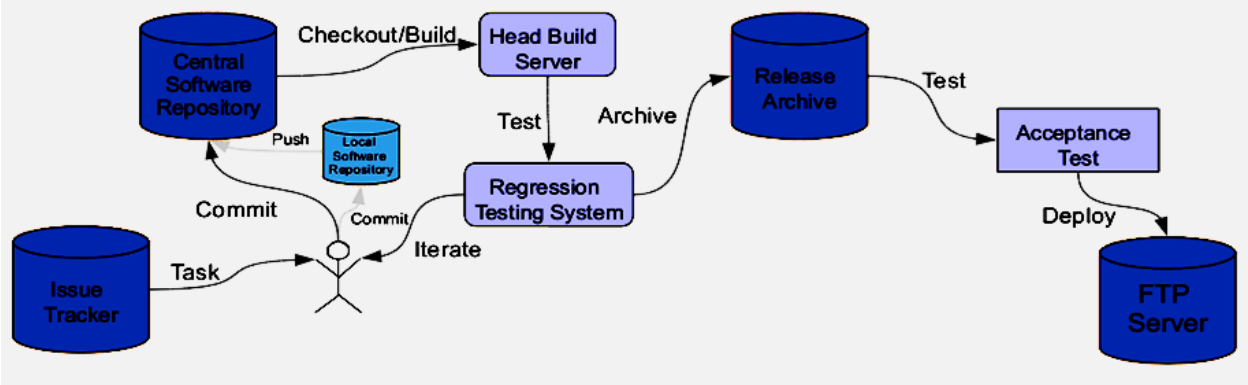

Fig. 3: Development process

Automated telephony testing can be integrated in this process. It will replace non-effective manual testing where some of the tests are not realizable. Automated telephony testing can be used for simulating for instance high frequency of calls, long duration calls, parallel combination of different calls and so on. It would be impossible to create for instance on thousand calls with length of one second and with pause between them also only one second. That's why automation of this testing is so important. It can create conditions which would be difficult or impossible to create manually.

In telephony, there is a lot of possibilities of testing. Some of them are listed below.

- Functional tests
    - Incoming/outgoing calls between analog/ISDN phones and VoIP
    - Calls with/without caller ID
    - Emergency calls - high priority
    - Frequency calls
    - Parallel calls
    - Calls with different duration
    - Combination of different call settings
    - Control of tones
- Control of information elements - for some supplementary services
    - Control of CGPN (Calling Party Number)
    - Control of CDPN (Called Party Number)
- Supplementary services
    - Calling Line Identification Presentation (CLIP)
    - Calling Line Identification Restriction (CLIR)
    - Advice of Charge (AOC)
    - Call Waiting (CW)
    - Call Forwarding (CFx)
    - Call Hold (HOLD)

3. Choosing Hardware

3.1 Important Properties of ISDN Cards

3.1.1 Service Levels

As already said there are two types of service levels – Basic rate interface and Primary rate interface. Basic rate interface has two B-channels and Primary rate interface has in Europe 30 B-channels. This makes PRI more interesting for e. g. large companies. More channels however means of course higher price on the market.

3.1.2 Number of Ports

Typical number of ports of ISDN cards differs from one port up to eight ports. Most of the cards of the market is equipped with two or four ISDN ports. There is also possibility to interconnect cards (from the same manufacturer) to get a higher number of the ports.

3.1.3 Port Configuration Protocol

ISDN cards can be configured to use one of port configuration protocols. First one is Point-to-Point Protocol (PPP) which means the communication is between two directly connected points in a network. The other one is Point-to-Multipoint Protocol (PMP) where communication offers several paths from single location to various locations – one-to-many.

3.1.4 Modes of Operation

The modes of operation basically copy some of ISDN devices mentioned above. The two possibilities are Terminal equipment mode and Network terminal mode. TE is equipment which complies with ISDN standards and NT creates physical and logical boundary between the customer's premises and the carrier's network. In practical use it means that NT mode is

for connecting ISDN telephones to the card and TE mode is for connecting for instance to a gateway.

3.1.5 Active/Passive Cards

Another important property of every ISDN card is whether it is active or passive card. Active cards have their own CPU and memory to handle the communication. Passive ones use CPU and memory of a computer to which they are connected. It is of course better to use active card because it does not stress a used PC. On the other hand the difference in the prices of active and passive cards is quite large.

3.1.6 Other Properties

There are also other properties of ISDN cards like support of Euro-ISDN stack, type of used bus (usually some version of PCI or USB) or integration of Echo cancellation module. Some of the cards also specifically support some software like mISDN drivers or some softswitches.

3.2 Market Research

For purposes of this thesis was necessary to buy an ISDN card. That is why a market research was needed. In (Tab. 2) there are various ISDN PCI cards which were chosen to be considered for telephony testing in Sphairon.

	Service level	Number of ports	Configurable PPP/PMP	Configurable TE/ NT mode	Active/Passive	Euro-ISDN	Bus	Echo cancellation module	Support	Price
Eicon Dialogic Diva 4BRI-8 [14]	BRI	4	Yes	Yes	Active	Yes	PCI 2.2	Yes	Asterisk	947 €
OpenVox B400P [15]	BRI	4	Yes	Yes	Passive	Yes	PCI 2.2	No	mISDN, Asterisk, FreeSwitch, Yate	290 €
OpenVox BE400E [15]	BRI	4	Yes	Yes	Passive	Yes	PCI Express 1.0	Yes	mISDN, Asterisk, FreeSwitch, Yate	393 €
OpenVox BE400P [15]	BRI	4	Yes	Yes	Passive	Yes	PCI 2.2	Yes	mISDN, Asterisk, FreeSwitch, Yate	391 €
OpenVox B400E [15]	BRI	4	Yes	Yes	Passive	Yes	PCI Express 1.0	No	mISDN, Asterisk, FreeSwitch, Yate	291 €
Junghanns quadBRI® 2.0 PCI ISDN [16]	BRI	4	Yes	Yes	Passive	Yes	PCI 2.2	No	mISDN, Asterisk	469 €
Digium B410P [17]	BRI	4	Yes	Yes	Passive	Yes	PCI 2.2	Yes	Asterisk	560 €
Sangoma A500 [18]	BRI	3	Yes	Yes	Passive	Yes	PCI 2.2/PCI Express	No	Asterisk, Freeswitch, Yate	238 €
Sangoma B500 [19]	BRI	4	Yes	Yes	Passive	Yes	PCI Express	No	Asterisk, Freeswitch, Yate	479 €
Beronet BN4S0 [20]	BRI	4	Yes	Yes	Passive	Yes	PCI 2.2/PCI Express	Yes	Asterisk, mISDN	492 €

Tab. 2: Comparison of ISDN cards

3.3 Decision

The market research was done with respect to requirements of Sphairon GmbH (a ZyXEL Company). That is why only cards with BRI ports are listed here. Current Sphairon products does not support Primary rate interface. Also 4 ports were needed which meant either one 4-port card or two 2-port cards. However, the second choice was not very expedient from the financial point of view. The next desired property was that for every port would be possible to configure its port configuration protocol PPP/PMP and its mode TE/NT. There was also important that the card would support Euro-ISDN protocol with its features (call waiting, call forwarding, advice of charge...).

According the desired features OpenVox B400E ISDN card was chosen. It fulfils all requirements and offers the best ration of price and power.

Fig. 4: OpenVox B400E ISDN card [21]

4. Implementation

4.1 Card Installation and Call Configuration

First part of an implementation of the ISDN card was to install all the necessary software requirements and use them to make simple calls between any softphone and an ISDN telephone connected to the ISDN card. The setup below is only to get a feeling about how the ISDN card works and which software configuration is needed to get the card working. The card itself does not come with any software which means that the user has to find out everything on his own. On the other hand it also means that the card should work with standard universal Linux drivers and software.

For creating the calls, the following is software was chosen:

- ISDN Card
- mISDN V2
- mISDNuser V2
- Linux Call Router
- Asterisk
- Softphone (ZoiPer)

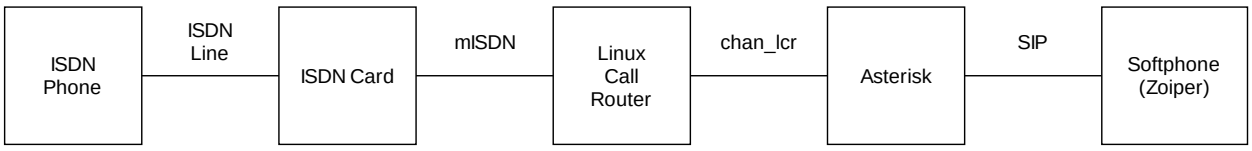

Fig. 5: Block diagram for used setup

When the call is initiated in the ISDN phone it goes through ISDN line to the ISDN PCI card. Then it continues to Linux Call Router which controls the card by mISDN drivers. Linux Call Router is connected to Asterisk by LCR channel and Asterisk connects the call to a softphone using Session Initiation Protocol. If the call is initiated in the softphone, it goes the other way around. All parts of the setup are described below.

4.1.1 ISDN Card Hardware Settings

The first step in the setup is hardware configuration of the ISDN card. Power feeding connector on the card has to be set to Enable/Disable (depending on used phone), the NT/TE settings on the card should be set to NT (for connection of an ISDN phone) and the termination on the card must be adjusted to ON. All these features are in the scheme of the card below.

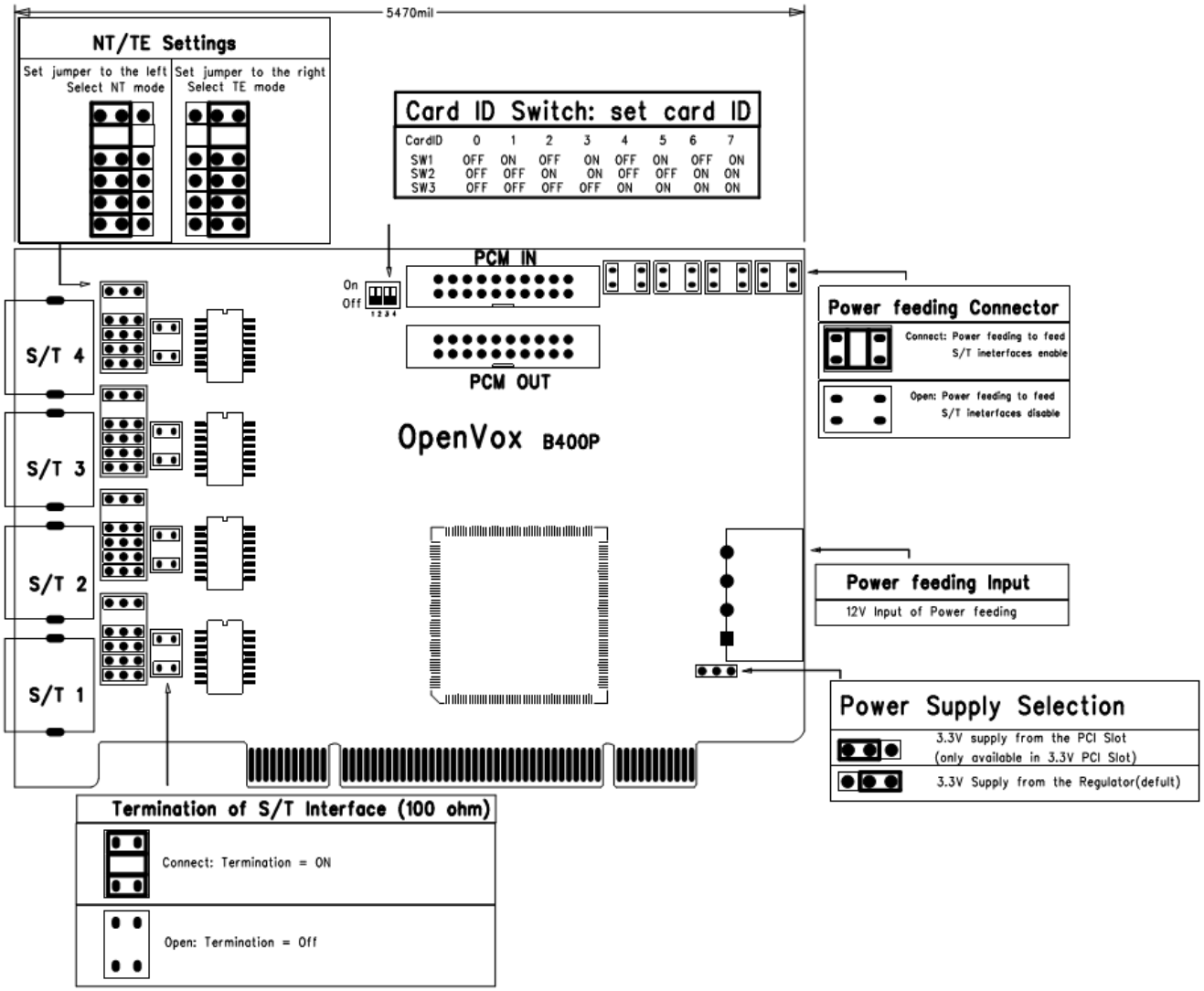

Fig. 6: ISDN card scheme [22]

4.1.2 mISDN

The mISDN [23] is a modular ISDN driver for Linux which supports various ISDN cards. Mostly it supports Cologne Chips Design HFC-PCI based cards. The mISDN consist of mISDN in the kernel space and mISDNuser in the user space.

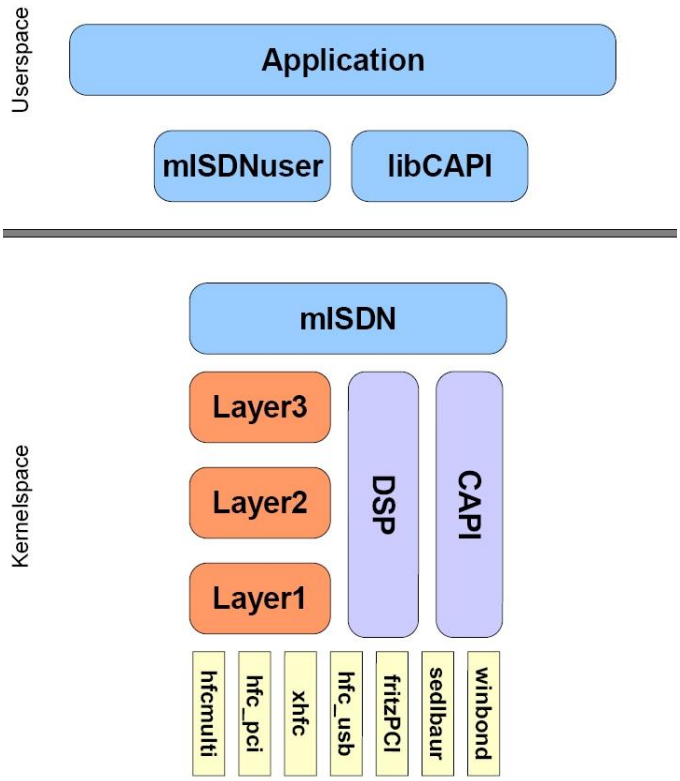

Fig. 7: *Architecture of mISDN [24]*

The task was done in Ubuntu 12.04 operating system with kernel 3.2.0-61-generic-pae which has mISDN v2 driver already included. The next step was to download and install mISDNuser. It was downloaded from mISDN git repository to ensure it is the latest version and then it was configured and installed by following commands in Linux shell terminal.

```
git clone git://git.misdn.eu/mISDNuser.git/
make
./configure
make
make install
```

Code 1: Installation of mISDNuser

4.1.3 Linux Call Router

Linux call router [25] is an ISDN call router. It is able to work with ISDN cards through mISDN driver which makes it very important part of this software configuration. The latest version of LCR (1.7) can be downloaded from mISDN git repository. Then the LCR is installed by the following commands. It has also very important feature to work with Asterisk by using module chan_lcr.so which is generated during the installation and copied to Asterisk-modules directory.

```
git clone git://git.misdn.eu/lcr.git/
./configure --with-asterisk
make
make install
cp chan_lcr.so /usr/lib/asterisk/modules/
```

Code 2: Installation of Linux Call Router

Installation of Linux call router also includes a tool which creates a shell script for start, stop and restart of mISDN. Start the tool by entering the following command and continue through all options.

```
genrc

This program generates a script, which is used to
start/stop/restart mISDN
driver. Please select card only once. Mode and options are
given by LCR.
Select driver for cards:
 (1) HFC PCI (Cologne Chip)
 (2) HFC-4S / HFC-8S / HFC-E1 (Cologne Chip)
 (3) HFC-S USB (Cologne Chip)
Select driver number[1-n] (or enter 'done'): 2
Select driver number[1-n] (or enter 'done'): done
Enter options of mISDN_dsp module. For a-LAW, just enter 0.
For u-LAW enter 1.24
[0..n | 0xn]: 0
Enter debugging flags mISDN core. For no debug, just enter 0.
[0..n | 0xn]: 0
Enter debugging flags of cards. For no debug, just enter 0.
[0..n | 0xn]: 0
Enter dsp debugging flags of driver. For no debug, just enter
0.
[0..n | 0xn]: 0
Enter location of the mISDN modules. Enter '0' for kernel's
default
location. Enter '1' for binary distribution's location
'/usr/local/pbx/modules' or enter full path to the modules
dir.
[0 | 1 | <path>]: 0
Finally tell me where to write the mISDN rc file.
Enter the name 'mISDN' for current directory.
You may want to say '/usr/local/lcr/mISDN' or
'/etc/rc.d/mISDN'
: mISDN
```

Code 3: Generating script for control of mISDN

After this, script with name "mISDN" is created. It can be used with parameters start/stop/restart/help.

```
sh mISDN start
sh mISDN stop
sh mISDN restart
sh mISDN help
```

Code 4: Control of mISDN

After starting the mISDN script, the proper loading of drivers can be checked by using *lsmod* command and LCR query command and the result should look like this:

```
lsmod

Module                  Size  Used by
hfcpci                 28300  0
mISDN_dsp             203600  0
mISDN_core             80396  17 mISDN_dsp,hfcpci

lcr query

LCR  Version 1.14
Using 'misdn_info'

Found 4 ports
Port  0 'hfc-4s.1-1': TE/NT-mode BRI S/T (for phone lines &
phones)
          2 B-channels: 1-2
            B-protocols: RAW HDLC X75slp L2:DSP L2:DSPHDLC
--------
Port  1 'hfc-4s.1-2': TE/NT-mode BRI S/T (for phone lines &
phones)
          2 B-channels: 1-2
            B-protocols: RAW HDLC X75slp L2:DSP L2:DSPHDLC
--------
Port  2 'hfc-4s.1-3': TE/NT-mode BRI S/T (for phone lines &
phones)
          2 B-channels: 1-2
            B-protocols: RAW HDLC X75slp L2:DSP L2:DSPHDLC
--------
Port  3 'hfc-4s.1-4': TE/NT-mode BRI S/T (for phone lines &
phones)
          2 B-channels: 1-2
            B-protocols: RAW HDLC X75slp L2:DSP L2:DSPHDLC
```

Code 5: Checking loading of the drivers

Configuration files of LCR are created in */usr/local/etc/lcr/* directory. The important ones for this application are *interface.conf* and *routing.conf*. In both these configuration files is possible to create various different settings which can all be found in [26]. Here is the used configuration of file *interface.conf*:

```
[ast]
remote asterisk
context from-lcr
earlyb yes
tones yes

[Int]
extension
msn 1234
portnum 1
bridge ast
nt
earlyb yes
tones yes
```

Code 6: LCR file interface.conf

The interface *ast* is for communication with Asterisk. The keyword *remote* sets Asterisk as the remote application and on the next line there is stated that context *from-lcr* in Asterisk configuration file should be used. The next two lines configure that this interface has to send and receive tones and announcements to and from all ports of the interface. The next interface *int* communicates with the ISDN card. This interface is internal, which is stated by the keyword *extension*. Only the number configured in the ISDN phone is allowed (in this case 1234). The interface uses port number 1, all calls are routed Asterisk and it runs in NT mode so the ISDN phone can be connected to the port. Setting of *routing.conf* is the following.

```
interface=ast    : intern
interface=Int    : extern interfaces=ast
```

Code 7: LCR file routing.conf

The used settings mean that calls from interface *ast* are forwarded to an internal extension and calls from interface *Int* are forwarded to an external interface *ast*.

The next step of LCR configuration is generating an extension. LCR has a command for that. It states the internal and external number of the used ISDN phone and interface which is used for it.

```
genextension 1234 Int 1234
```

Code 8: Generating an extension 1234 for interface Int

The Linux Call Router can be started in normal mode or as a daemon. There is also useful command to display information of the running instance of LCR and its log.

```
lcr start
lcr fork
lcradmin state
```

Code 9: Commands for using LCR

4.1.4 Asterisk

Asterisk [6] is one of the tested softswitches and due to its advantages (Tab. 1) and the possibility to be connected to Linux Call Router it was chosen for this task. It is used as a softswitch for this basic setting. Asterisk can be downloaded from *http://www.asterisk.org/downloads/* and installed by the following commands.

```
./configure
make
make install
make samples
make config
make install-logrotate
```

Code 10: Installation of Asterisk

The Asterisk configuration files are located in */etc/asterisk/*. There are two files which has to be modified. The first one is *sip.conf*. Here the numbers of SIP softphones are registered and their handling configured. In this case, there are two parts. The *general* part is used when there is no other match. *Context* defines part of a dialplan which is used. The 6001 is number of used softphone, *type* sets whether the context is used for inbound or outbound calls or both. The address of the phone in network is dynamically found and the password is set as "password". The last two lines are for resetting previous codec settings and configuring new ones.

```
[general]
context=default

[6001]
type=friend
context=default
host=dynamic
secret=password
disallow=all
allow=ulaw
```

Code 11: Asterisk file sip.conf

The second important file is so called dialplan in *extensions.conf*. There are two contexts used in the extensions file. The first one is *default* and it is used for the calls from softphone to the ISDN phone. When the 1234 extension is dialed, the call is connected through LCR using interface *ast* with identifier 1234 and timeout 20 s. The context *from-lcr* is used for calls issued from ISDN phone via LCR. There are two possibilities here configured. When the number 100 is used, the call is answered, then the message hello-world is played and the call is hanged. When the extension 6001 is dialed, the call is connected to the softphone through SIP channel.

```
[default]
exten = 1234,1,Dial(LCR/ast/1234,20)

[from-lcr]
exten = 100,1,Answer()
same = n,Wait(1)
same = n,Playback(hello-world)
same = n,Hangup()
exten = 6001,1,Dial(SIP/6001,20)
```

Code 12: Asterisk file extensions.conf

The last step of Asterisk configuration is starting the Asterisk and loading the channel for communicating with Linux Call Router. The Asterisk here is started with level 5 of verbosity and debug for getting log messages that help to get information about calls and later on are used to evaluate the calls.

```
asterisk –cvvvvvddddd
module load chan_lcr.so
```

Code 13: Starting Asterisk and loading chan_lcr module

4.1.5 Softphone (ZoiPer)

ZoiPer [27] is free VoIP softphone which uses Session Initiation Protocol. Of course, any other similar softphone can be used for this purpose. It can be downloaded, untared and started by:

```
wget http://www.zoiper.com/downloads/free/linux/zoiper219-
linux.tar.gz
tar -xvz zoiper219-linux.tar
./zoiper
```

Code 14: Installation and start of ZoiPer

The configuration can be done in seven steps:

- Click on options
- Add new SIP account
- Enter chosen number of your softphone (6001) for the account name => OK
- Enter the IP address of your Asterisk system in the Domain field
- Enter chosen number of your softphone (6001) in the Username field
- Enter your SIP peer's password (password) in the Password field
- Enter whatever you like in Caller ID Name or leave it blank

4.2 Test Setup Configuration

The requirements on the automated telephony testing are the following:

- Testing of outgoing calls
- Testing of incoming calls
- Testing of more parallel calls at the time
- Testing of calls with high frequency
- Testing of long-duration calls

In (Fig. 8) there is a block diagram of a setup used for automated testing. The calls are initiated from the Asterisk in test PC which works here as a dialer. If the call is set as outgoing, it goes through LCR channel to Linux Call Router, then to the ISDN PCI card using mISDN drivers. The ISDN card is connected to the ISDN ports of the Device Under Test (DUT) by ISDN cables. The number of ISDN ports depends on the DUT. The call is then handled by the DUT and routed according a configuration (below) to a softswitch (Asterisk-server) where it is answered. Thus the setup acts like when the call is initiated from an inner telephony network to the outside world.

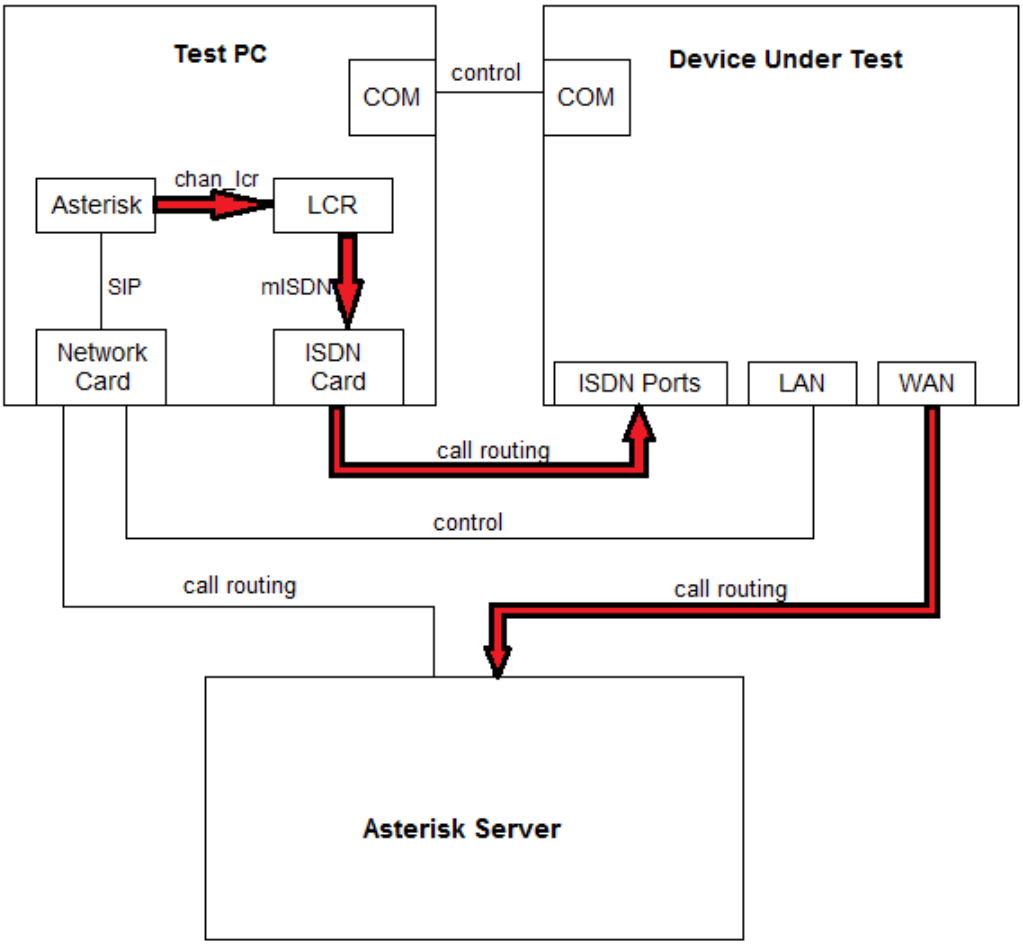

Fig. 8: Block diagram of outgoing calls simulation for automated testing

If the call is set as incoming, it uses SIP channel and goes to the Asterisk Server by LAN connection. In the server the call is routed to the DUT and then through ISDN card and LCR to Asterisk where it is answered. The Asterisk Server represents again the telephony provider so the setup simulates an incoming call. The connections by LAN and serial port between test PC and DUT are used for control of the DUT.

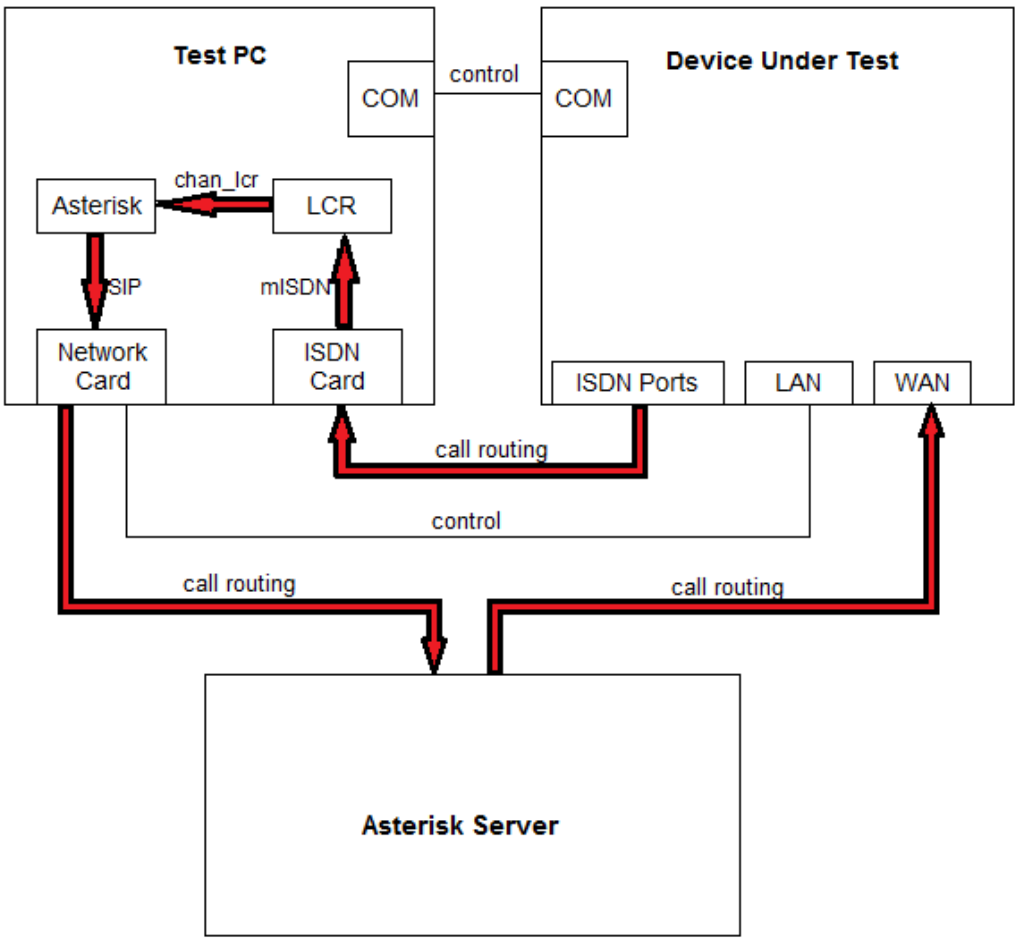

Fig. 9: Block diagram of incoming calls simulation for automated testing

4.2.1 Asterisk Server Configuration

The first step in automating telephony testing was configuring an Asterisk Server. For the purposes of the testing a virtual machine based on Debian 7 and accessible in the Sphairon network was set. The server can be accessed and controlled by SSH connection. An Asterisk instance was installed on this virtual machine with configuration for answering calls with numbers used for outgoing calls, routing calls with numbers used for incoming calls and registering extensions which are set in DUT. Examples of the setting are shown in the following codes. The time for the Wait application is here set to practically infinite so the calls can be of arbitrary length and they will be always hanged up by the initiating side.

```
[default]
exten = 990000,1,Dial(SIP/990000,20)
exten = 880000,1,Answer()
same = n,Wait(99999999)
same = n,Hangup()
```

Code 15: Example of extensions.conf in Asterisk Server

```
[general]
context=default

[990000]
type=friend
context=default
host=dynamic
secret=990000
disallow=all
allow=ulaw
```

Code 16: Example of sip.conf in Asterisk Server

4.2.2 Test PC Configuration

The configuration of the test PC is similar to the previous configuration for basic calls. The ports of the ISDN card need to be set to TE mode and the power feeding should be disabled. The same software is needed. In Asterisk, there is a possibility to create automated calls by creating a call file in defined structure and the call is initiated by moving it to Asterisk outgoing directory */var/spool/asterisk/outgoing/*. The call file is located in the same directory as the test script.

```
Channel: LCR/ast/<called number>
CallerID: <caller number>
Application: Wait
Data: 1
```

Code 17: Structure of used call file

In the first line, there is set that channel LCR is used and in Linux Call Router context *ast*. Then of course the called number is necessary. *CallerID* sets the number of the caller. Next lines set what happens after the call is answered. In this case application Wait is started which

just sticks on the line for the defined number of seconds (in this example 1 second) and then it hangs up.

The first issue here is that calls which are initiated in Asterisk and go through chan_lcr to Linux Call Router do not have any caller ID. It is probably due a bug in the LCR channel so a way around this had to be found. In LCR there is a possibility to map call numbers going through. So to each port was assigned a number to be set which corresponds to the setting of the DUT.

```
[ast]
remote asterisk
context from-lcr
earlyb yes
tones yes

[te-mode0]
portnum 0
screen-out % unknown present 990000%
[te-mode1]
portnum 1
screen-out % unknown present 991111%
[te-mode2]
portnum 2
screen-out % unknown present 992222%
[te-mode3]
portnum 3
screen-out % unknown present 993333%
```

File interfaces.conf in test PC

```
[main]
interface=ast                 : intern
interface="te-mode0"          : extern interfaces=ast
interface="te-mode1"          : extern interfaces=ast
interface="te-mode2"          : extern interfaces=ast
interface="te-mode3"          : extern interfaces=ast
```

File routing.conf in test PC

All the necessary Asterisk configuration files are handled in the script so the last thing to configure is reading log messages from the DUT. It is done using syslog-ng Ubuntu package. After installation of this package a configuration file has to be created in */etc/syslog-ng/conf.d/*.

```
source s_udp {
    udp(port(514));
};

destination df_sphairon {
    file("/var/log/sphairon.log");
};

destination df_dut_log {
    file("/var/log/dut_log.log");
};

filter f_sphairon {
    host("192.168.100.*");
};

log {
    source(s_udp);
    filter(f_sphairon);
    destination(df_sphairon);
    destination(df_dut_log);
};
```

Code 18: Syslog configuration file

After creating the file, the syslog service must be restarted to get the log messages from the DUT. The log messages are saved into two files *sphairon.log* and *dut_log.log*. The second one is only for usage of the script and gets cleaned at start of every test.

```
/etc/init.d/syslog-ng restart
```

Code 19: Restart of the syslog service

For controlling the DUT via serial port there is a software called ckermit. It has to be installed on the test PC and file *~/kermrc.ttyS0* with the following structure must be created.

```
set line /dev/ttyS0
set speed 115200
set carrier-watch off
set handshake none
set flow-control none
set prompt {kermit-ttyS0> }
log session ~/kermlog.ttyS0
connect
```

Code 20: File kermrc.ttyS0

The remote access is started by:

```
kermit ~/kermrc.ttyS0
```

Code 21: Starting ckermit

4.2.3 DUT Configuration

There are currently two devices to be tested – Speedlink 5501 and Gateway 400 dp NC. In the first one, there is only one ISDN port. In the second device, there are four ISDN ports. Beside that the configuration is pretty much the same. It is done in a Web Interface. At first a WAN setup has to be performed. This is done in tested devices automatically. After that a VoIP provider needs to be configured.

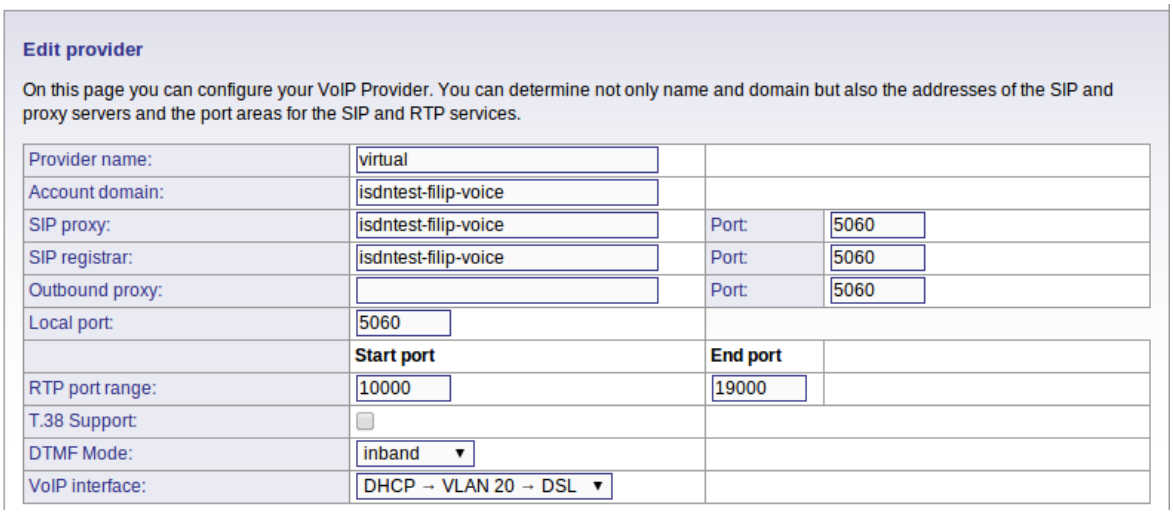

Fig. 10: Configuration of VoIP provider

Then all used VoIP accounts must be set. These accounts are the same which are registered in the Asterisk Server.

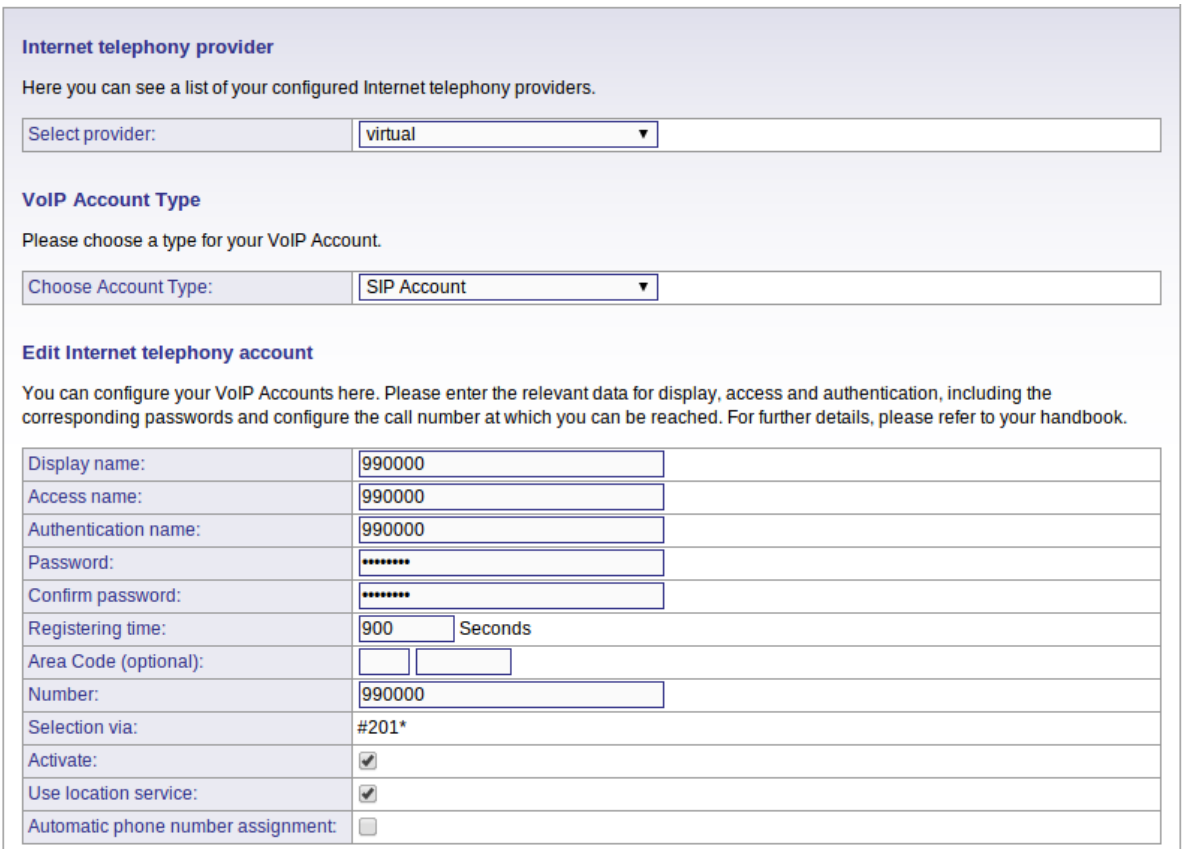

Fig. 11: Configuration of VoIP account

The next part (only for SIP Gateway) is setting of ISDN interfaces in the ISDN section.

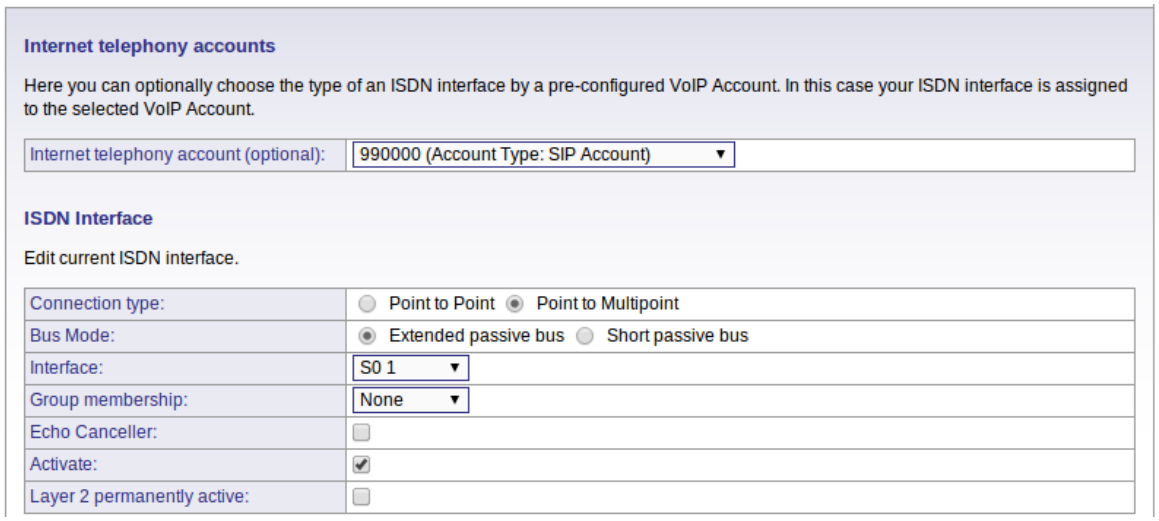

Fig. 12: Configuration of ISDN interface

In the remote control was extended level of debug messages for better analysing of results and a remote syslog was enabled.

```
voip /tmp/voip_socket tr change CallCtrl 1023
voip /tmp/voip_socket tr change ExosipCtrl 1023
voip /tmp/voip_socket tr change DspApi 1023
voip /tmp/voip_socket tr save
cfgclient "updatekey Syslog Id 1 EnableRemoteLogging \
integer:1 RemoteLoghost text:192.168.100.100;"
```

Code 22: Commands used in remote access to the DUT

4.3 Automated Testing

The automation of the testing was achieved by creating a program in shell script. The shell script is designed to work with a Linux system. It is quite easy to control programs, processes and files in Linux by using shell script language. The program creates automated calls in given length with defined properties. These properties are set before the test in a configuration file *teltest.conf*. The program is divided into two scripts. There is a main script *start_teltest.sh* and it controls another script *call_daemon.sh* which as the name implies runs in the background as a daemon. The second one originates the calls and checks the results which are returned using a save file back to the main script. It runs as a daemon because there can be up to eight parallel instances running and creating eight parallel calls.

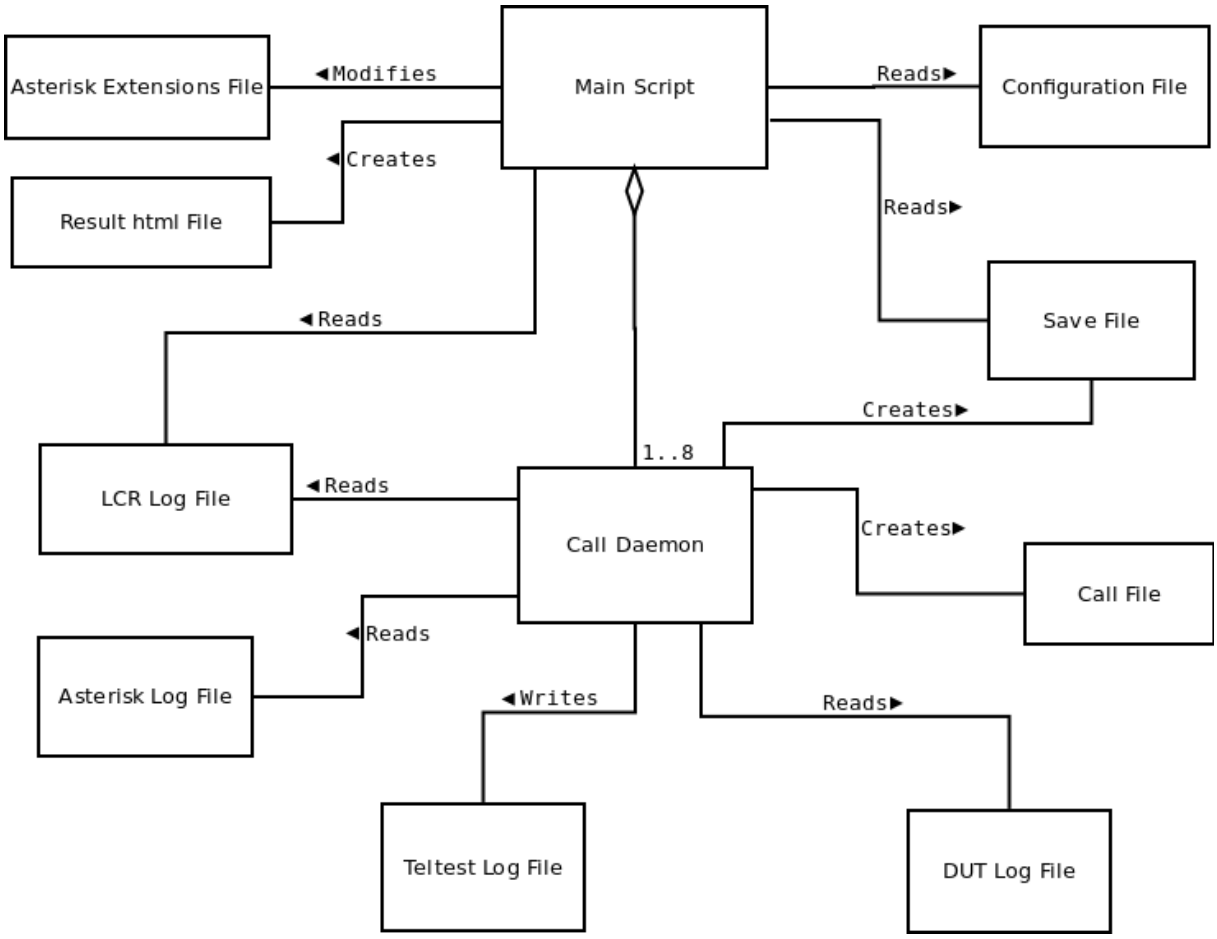

Fig. 13: Software design diagram

4.3.1 Configuration File

Inside the configuration file, user can set an arbitrary test. The first parameter is a caller number. This number must correspond with the setting of Linux Call Router and the DUT in order that the caller number works properly. The same thing applies for the second parameter which is a called number. There must also be set a number of calls with given configuration, length of the calls and pause between the calls. These two specify frequency with which these calls are originated. Then the user can choose if incoming or outgoing calls should be simulated. The last parameter is number of the group. The idea here is that quite complicated test can be configured and executed in groups. In each group can be different number of parallel calls with different parameters.

In the configuration file below is an example of a configuration of a test. In the first group, there are four parallel outgoing calls with high frequency. Every call has a length of one second and pause of one second. These calls are executed one thousand times. In the group number 2, there are two parallel incoming calls of length 10 seconds and pause 5 seconds. They are created 50 times. In the last group, there are two long-term calls. Both are executed only ones but they last for ten hours. One of them is set as incoming, the other one as outgoing.

```
#Configuration file for script start_teltest.sh
#
#This file needs to be in the same directory as
start_teltest.sh.
#Set up your telephony test here.
#
#Syntax is:
#<caller number> <called number> <number of calls>
#<length of calls> <pause between calls> <in/out>
#<group of prallel calls>
#
#Use white space between parameters.
#Time parameters are in seconds
#in/out stands for incoming/outgoing calls
#
#Example:
#470000 471111 500 2 0.5 out 1

990000 880000 1000 1 1 out 1
991111 881111 1000 1 1 out 1
992222 882222 1000 1 1 out 1
993333 883333 1000 1 1 out 1

880000 990000 50 10 5 in 2
881111 991111 50 10 5 in 2

990000 880000 1 36000 1 out 3
881111 991111 1 36000 1 in 3
```

Code 23: Configuration file teltest.conf

4.3.2 Main Script

The job of the main script *start_teltest.sh* is to read the configuration file and to create a test according it. Also to start given number of instances of the call daemon, wait for them to finish and at the end export results from the test in html file. In the following figure is a flowchart which shows the function of the main script.

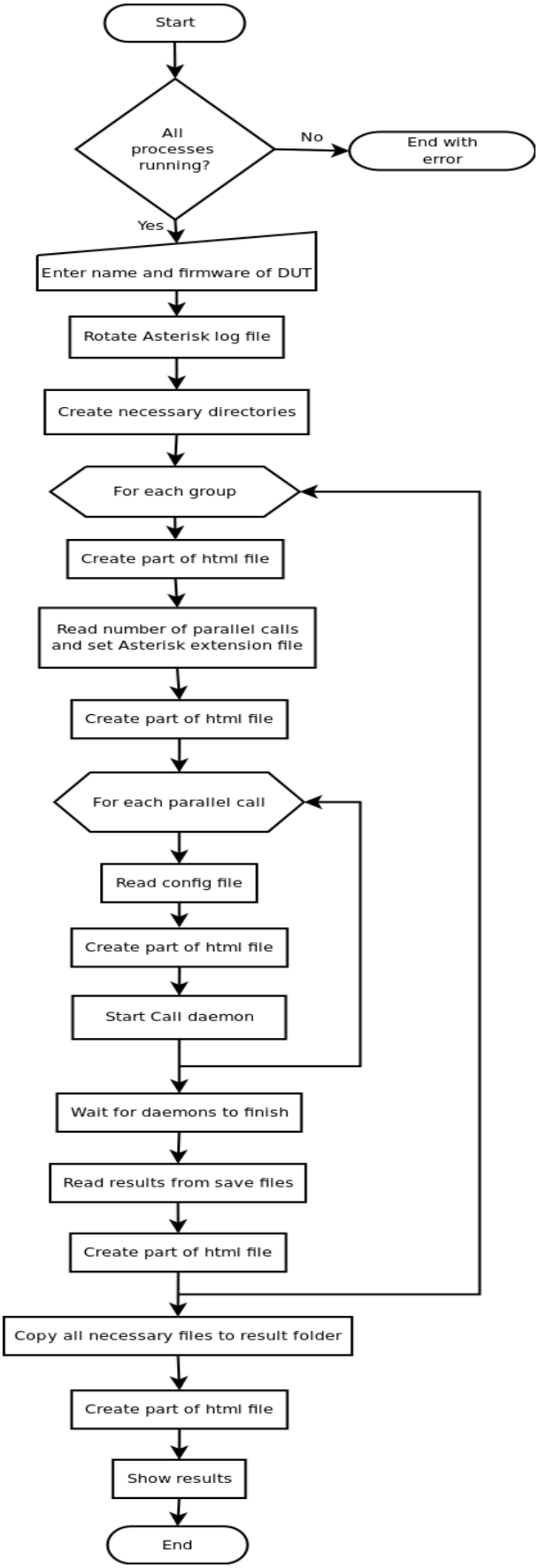

Fig. 14: Flowchart describing function of the main script

At start the program calls the *CheckProcess* function so the necessary processes Asterisk and Linux Call Router are running.

```
CheckProcess()
{
     processNum=`ps aux | grep $1`
     if [ "$processNum" = "0" ]
     then
          echo "$1 is not running!"
          exit 1
     fi
}

CheckProcess "asterisk"
CheckProcess "lcr
```

Code 24: Checking necessary processes

If the test in manually stopped by *Ctrl+C* the action is detected and function *Stop* is called which stops all the running subprocesses and the main script finishes all the necessary things before stopping.

```
Stop()
{
     toKill=$(ps aux |grep "call_daemon.sh" |grep -v grep
|awk '{print $2}')
     kill $toKill
}

trap "Stop" 2
```

Code 25: "Graceful" stopping

In the next step the program asks the user to enter a name and firmware version of the tested device. This is saved into a save file so if the DUT has not changed it is possible to leave the field blank and the previous information is used. According the information a directory with DUT name, firmware version and timestamp is created (if does not already exist). Then Asterisk log file is rotated. The log file is used to get information about the calls so if it would be large it would make the test much slower.

The results are exported into html file which uses JavaScript application Google Charts to visualize the results in pie charts. The html file is created during the test. Besides the charts it contains tables with information about the test.

After this first part, there is the main cycle which is executed for each group. Asterisk configuration file *extensions.conf* is modified according information read from the *teltest.conf*. The comments and empty lines are detected using command *egrep* and then skipped. For the incoming calls, there is set that the call to the given number should be answered, then Asterisk waits for the defined length of the call and after that the call is hanged up. Asterisk have to reload the extensions file in order to take effect of the changes.

In the next phase all of the parameters from the configuration file for the test are loaded and the subscript *call_daemon.sh* is started for every parallel call. The subscript is started with necessary parameters which have to be hand over to it.

When the subscripts are started, the main script is waiting for them to finish. If the subscript finishes properly as planned, it returns an error status which equals to zero. If it is stopped prematurely the error status is non-zero and the script does not start a new cycle.

After the subscripts are finished, the results are passed to the main script through temporary save files. The main script extracts them from the files. Successful and failed calls are counted as well as crashes of the DUT that can also lead to crash of the test PC.

Once the whole test is finished file with messages from the DUT is copied to a directory with timestamp of the test along with all the other files important for analyzing of the test. Then results are calculated and shown in html file.

4.3.3 Call Daemon

File *call_daemon.sh* serves for initiating and evaluating of the calls. It is started by the main script and it runs in a background. During the tests, there can be up to eight instances running simultaneously creating eight parallel calls at the same time.

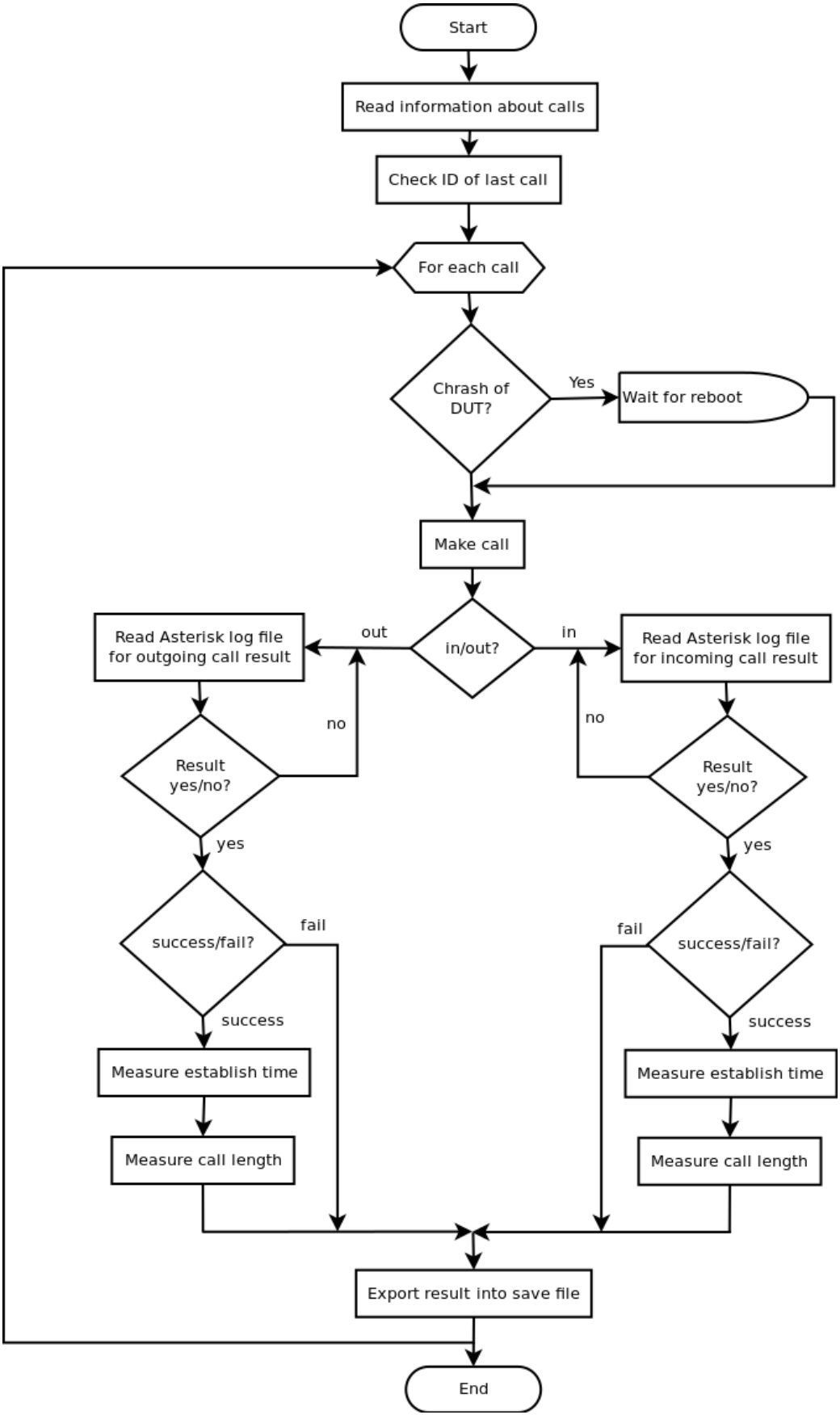

Fig. 15: Flowchart describing function of the call daemon

One of the functions of the call daemon is to create a log from the test. There is a function *ExportToLog* which takes care of that. The function uses a timestamp along with all the parameters and identification number of the call. The identification number has a format number of call/number of parallel call/number of group. After all these information the function adds a current status, which is passed as parameter of the function.

```
ExportToLog()
{
     currentTime=`date +"%F_%H:%M:%S"`
     echo "Call: $i/$parallelNum/$group [$currentTime] $inOut
from: $fromNumber to: $toNumber length: $lengthOfCalls s\
 pause: $pauseBetweenCalls s Status: $1" | tee -a $outputFile
     return
}
```

Code 26: Exporting messages into log file

The script at first checks ID of last call. It finds the last call to given number and its identification numbers so they can be compared with found messages in the Asterisk log file during a call. Thus is secured that script evaluates the right call.

The next important feature of the script is checking for crash of the DUT. The crash can be caused by the stress testing and when not handled it can bring about a crash of test PC. That's why a log file from the DUT is checked for panic messages.

```
CheckDUT()
{
     #Check state of DUT
     dutLog=$(tail -500 $dutLogFile)
     if echo "$dutLog" |grep -q "Fatal exception: panic"
     then
          currentTime=`date +"%F_%H:%M:%S"`
          echo "Error [$currentTime] DUT crash" \
          |tee -a $outputFile
          echo "DUT reboots, waiting..."
          errors=$((errors + 1))
          sleep 120
          WaitForReboot
     fi
}
```

Code 27: Checking DUT for panic messages

If the panic occurs, it is detected and a function *WaitForReboot* is called. It waits for the DUT to reboot and checks the DUT log file for messages of registering SIP accounts after reboot.

```
WaitForReboot()
{
     while true
     do
          dutLog=$(tail -200 $dutLogFile)
          echo "$dutLog" |grep -q \
          "User is successfully registred" && break
          sleep 1
     done
     sleep 1
}
```

Code 28: Waiting for reboot of DUT

The call is initiated by copying a template file *call_file.call.source* to a local call file, which is moved to Asterisk outgoing directory */var/spool/asterisk/outgoing/*. Before the call file is moved there, it is modified according settings of the call. One reason, why the file is copied at first is that only one template file is needed. Second reason is that the call file needs to be moved and not copied to the Asterisk outgoing directory. When a file is copied in Ubuntu, the copy appears in the target folder sequentially but if it is moved, only a pointer on that file is changed and the path of the file is changed at once.

After the call is initiated, measuring of an establish time is started and one of the functions for checking result of the call is originated. Functions *CheckIncomingResult* and *CheckOutgoingResult* continuously read log messages from asterisk and thus evaluate current status of an initiated call. As the names imply one of them is for incoming and the other one for outgoing calls. When the call is successfully established, the establish time is measured as well as length of the call. The length of the call must be the same as defined in the configuration file otherwise the call is evaluated as unsuccessful.

The last job of the call daemon is to export results into a file. Thus all the necessary information are passed back to the main script.

4.4 Results

Results of the tests are analysed and if there is any error a source of this error has to be found. There can be several causes. The test system is developed to discover bugs in tested software which is the first possible cause. But there can also be a bug in any part of the test setup (the used software, the test script). So each of the results have to be carefully examined. For the examination are used log files from the DUT, Asterisk and the test script. If a bug of the test setup is found it is fixed and the test is started again. If a bug of the DUT is discovered, it has to be further inspected and the tested software version gets back to a development process (Fig. 3). When the bug is fixed the software is tested again. Thus the quality of developed product is ensured.

The results of each test are exported into an html file. The html file uses JavaScript application Google Charts to create charts for each tested group and for the whole test. Besides the charts, there are also tables with configuration of the test. The results are furthermore saved into a text file in a simple form.

In addition to result files, there is also a file with log messages which contains results of the whole test as well. Examples of the log file, result html file and result test file are shown below.

```
Call: 1/1/1 [2014-06-25_15:43:22] out from: 990000 to: 880000
length: 1 s pause: 1 s Status: Making attempt
Call: 1/1/1 [2014-06-25_15:43:22] out from: 990000 to: 880000
length: 1 s pause: 1 s Status: Making a call
Call: 1/1/1 [2014-06-25_15:43:25] out from: 990000 to: 880000
length: 1 s pause: 1 s Status: Call failed
Call: 1/1/1 [2014-06-25_15:43:35] out from: 990000 to: 880000
length: 1 s pause: 1 s Status: Call instance freed
Call: 2/1/1 [2014-06-25_15:43:37] out from: 990000 to: 880000
length: 1 s pause: 1 s Status: Making attempt
Call: 2/1/1 [2014-06-25_15:43:37] out from: 990000 to: 880000
length: 1 s pause: 1 s Status: Making a call
Call: 2/1/1 [2014-06-25_15:43:38] out from: 990000 to: 880000
length: 1 s pause: 1 s Status: Call successfully established
Call: 2/1/1 [2014-06-25_15:43:39] out from: 990000 to: 880000
length: 1 s pause: 1 s Status: Correct call length
Call: 2/1/1 [2014-06-25_15:43:39] out from: 990000 to: 880000
length: 1 s pause: 1 s Status: Call instance freed

Test started at: 2014-06-25_13-53-23 on device: Gateway 400
dp NC 4.38.2.1.r79486
 2 calls were made in: 0:0:15
 Successful: 1
 Failed: 1
 Success: 50.00 %
 Crashes: 0
```

Code 29: Example of a log file

```
Group=1: Success=99 Failed=1
Group=2: Success=200 Failed=0
Group=3: Success=300 Failed=0
Group=4: Success=400 Failed=0
Group=5: Success=499 Failed=1
Group=6: Success=600 Failed=0
```

Code 30: Example of a result text file

Test started at: 2014-06-25_16-36-01 on device: Gateway 400 dp NC 4.38.2.1.r79486

Group 1

Caller Number	Callee Number	Number of Calls	Length of Calls	Pause Between Calls	Incoming/Outgoing	Average Establish Time
880000	990000	200	30	10	in	1.2
881111	990000	200	60	5	out	1.3
882222	991111	200	20	10	in	1.3
883333	991111	200	10	20	out	1.3
884444	992222	200	5	40	in	1.4
Successful=976 Failed=24 Average Establish Time=1.3						

Results

Successful Calls
Failed Calls

97.6%

Group 2

Caller Number	Callee Number	Number of Calls	Length of Calls	Pause Between Calls	Incoming/Outgoing	Average Establish Time
880000	990000	300	10	5	out	1.2
881111	990000	150	20	5	in	1.3
882222	991111	500	5	5	out	1.3
Successful=900 Failed=50 Average Establish Time=1.3						

Results

Successful Calls
Failed Calls

94.7%

Results

Successful Calls
Failed Calls

96.2%

Successful Calls
1876 (96.2%)

Fig. 16: Example of a result html file

Conclusion

The goal of this thesis was to develop a system which would automate a telephony testing of devices produced by Sphairon GmbH (a ZyXEL Company). The testing of telephony module in these devices was done manually which is very ineffective way. Also only some tests can be made in this way. The aim here was to make test of the devices for determination of their stability in unusually intense conditions – stress test.

The automated system for telephony testing developed during this thesis is able to automatically create automated calls. It uses several combination of software which is specifically configured so they all work together. It is capable of creating the calls in parallel so all of the eight channels in used device under test are used at the same time. It can also simulate incoming calls from DSL connection as well as outgoing calls from devices connected to ISDN ports of the device under test. Furthermore there is a possibility to configure parameters of each test like caller and called number, length and number of configured calls, pause between them and if the call should be incoming or outgoing. Besides that, really complicated tests with different parameters can be set in groups in one configuration file.

The produced testing system can stress the devices under test by automatically originated calls in high frequency (minimal length of a call is one second) and it can create call of practically unlimited length as well. Of course there is a possibility of any combination of calls with different settings so during one test there can be several long-term calls and several calls with high frequency which helps stress the tested devices. This would be never possible if only manual testing is used.

Thanks to all of these properties the developed automated test system already helped to discover some issues and bugs in tested software versions of the devices under test. It confirms that the product of this thesis is useful and helpful in the process of a regression testing in Sphairon GmbH (a ZyXEL Company).

In future the produced automated telephony test is supposed to be included into complex test process in Sphairon. It will be part of a regression testing system to ensure even better quality of produced devices. Also it would be useful to “simplify” the current setup to use only mISDN drivers in the test PC and not Asterisk and Linux Call Router. In every software a bug can appear so the less software is used the lower is probability of issues on the test PC side.

Resources

[1] ROUSE, M.: *What is signalling? – Definition from WhatIs.com.* [online]. [cit. 2014-04-11]. <http://whatis.techtarget.com/definition/signaling>

[2] Dialogic Corporation: *Telephony Fundamentals an Introduction to Basic Telephony Concepts*. Montreal, Quebec, 2007. 11 Pgs.

[3] BECKER, R.: *ISDN Tutorial.* [online]. [cit. 2014-04-30]. <http://www.ralphb.net/ISDN/>

[4] VALDES, R. and ROOS, D.: *How VoIP Works.* [online]. [cit. 2014-05-02]. <http://computer.howstuffworks.com/ip-telephony.htm>

[5] Rosenberg, J.: SIP: *Session Initiation Protocol.* , RFC 3261, June 2002. 269 Pgs.

[6] *Asterisk.org* [online]. [cit. 2014-07-16]. <http://www.asterisk.org>

[7] *FreeSWITCH | Communication Consolidation* [online]. [cit. 2014-07-16]. <http://www.freeswitch.org>

[8] *Yate* [online]. [cit. 2014-07-16]. <http://yate.null.ro>

[9] *Gemeinschaft 5.1* [online]. [cit. 2014-07-16]. <http://amooma.de/gemeinschaft/gs5>

[10] *Adore SoftSwitch* [online]. [cit. 2014-07-16]. <http://www.adoreinfotech.com/softswitch.html>

[11] *ControlSwitch System* [online]. [cit. 2017-07-16]. <http://www.dialogic.com/en/products/softswitch/controlswitch-system.aspx>

[12] *Cirpack* [online]. [cit. 2014-07-16]. <http://www.cirpack.com>

[13] Rosenberg, J.: *Session Traversal Utilities for NAT (STUN).* , RFC 5389, October 2008. 51 Pgs.

[14] *Diva 4BRI-8 | Dialogic* [online]. [cit. 2014-07-16]. <www.dialogic.com/en/products/media/diva/diva-4bri-8.aspx>

[15] *ISDN BRI Cards* [online]. [cit. 2014-07-16]. <www.openvox.cn/en/products/telephony-cards/isdn-bri-cards.html>

[16] *quadBRI 2.0 PCI ISDN* [online]. [cit. 2014-07-16]. <www.junghanns.net/de/quadBRI2_produkt.html>

[17] *Digital Euro ISDN BRI Cards | Digium* [online]. [cit. 2014-07-16]. <www.digium.com/en/products/telephony-cards/digital/euro-isdn-bri>

[18] *A500 2-24 port Scalable S/T BRI | Sangoma* [online]. [cit. 2014-07-16]. <www.sangoma.com/products/a500-2-24-port-scalable-st-bri/>

[19] *Sangoma B500 4xBRI/S0 PCIe Card* [online]. [cit. 2014-07-16]. <www.lieske-elektronik.com/product_sangoma-b502e-b500-4xbri-s0-pcie-card__487824.htm>

[20] *The beroNet Baseboard PCI or PCIe –beroNet GmbH* [online]. [cit. 2014-07-16]. <www.beronet.com/product/beronet-cards>

[21] CTI-PRO, s.r.o.: *OpenVox B400E - 4 port ISDN BRI PCIe card.* [online]. [cit. 2014-06-06]. <http://shop.ctipro.cz/lang-cs/2618-openvox-b400e-4-port-isdn-bri-pcie-card-.html&curr=6>

[22] *OpenVox B200P B400P B400E User Manual*. September 2007. 11 Pgs.

[23] *mISDN* [online]. [cit. 2014-06-06]. < https://www.misdn.eu>

[24] *About mISDN - MISDN.org* [online]. [cit. 2014-06-06]. <https://www.misdn.eu/wiki/About_mISDN/>

[25] Eversberg, A.: *Linux-Call-Router* [online]. [cit. 2014-06-06]. <http://www.linux-call-router.de>

[26] Eversberg, A.: *Linux-Call-Router, Software based ISDN Private Branch Exchange for Linux 1.2.* 2004. 105 Pgs.

Glossary

Signalling	Information for maintaining a telephone call.
In-band signalling	Signalling in the same channel as a telephone call is.
Out-of-band signalling	Signalling in its own separated channel.
Private Branch Exchange	Device for switching telephone calls.
Integrated Services Digital Network	Digital telephony system.
ISDN Service Levels	Types of ISDN ports (BRI/PRI).
Voice over Internet Protocol	Telephony which uses internet network for transmitting a digitalized voice.
Session Initiation Protocol	Application-layer control protocol used for controlling real-time multimedia sessions.
Softswitch	Device which connects telephone calls in a telecommunications network.
Asterisk	One of the most used softswitches.
mISDN	Universal Linux driver for ISDN cards.
Linux Call Router	Software engine for routing calls to/from ISDN card.
Terminal equipment mode	Mode of ISDN port for connecting gateways.
Network termination mode	Mode of ISDN port for connecting telephones.
Extension	Additional telephone connected to a telephone line.
Dialer	Device which automatically generates calls.
Call file	File with defined structure which after moving into a special Asterisk directory generates a call.
Daemon	Process that runs at the background

A Main Script

```
#!/bin/bash
#
#Script for making repeted calls using Asterisk, Linux Call
#Router and ISDN card
#
#Script uses call_file.call.source in the same directory
#which has to be configured

#Pathes to used files
dutLogFile="/var/log/dut_log.log"
lcrLogFile="/usr/local/var/log/lcr/log"
asteriskExtensionsFile="/etc/asterisk/extensions.conf"
configFile="teltest.conf"
saveFile="saveFile"

#Default values
successfulCalls=0
failedCalls=0
errors=0
finished=0
error="false"


CheckProcess()
{
     processNum=`ps aux | grep $1`
     if [ "$processNum" = "0" ]
     then
          echo "$1 is not running!"
          exit 1
     fi
}

Stop()
{
     toKill=$(ps aux |grep "call_daemon.sh" \
     |grep -v grep |awk '{print $2}')
     error="true"
     kill -13 $toKill
     ReleaseCalls
}

ReleaseCalls()
{
     for k in `seq 1 $((parallelCalls * 40))`
     do
          lcrLines=$(tail -$k $lcrLogFile)
```

```
          if echo "$lcrLines" \
          |grep -q "EP(.*): SETUP .* CH(.*)  interface"
          then
               endPoint=$(echo "$lcrLines" \
               |grep "EP(.*): SETUP .* CH(.*)  interface")
               endPoint=$(echo ${endPoint#*EP\(})
               endPoint=$(echo ${endPoint%\)\: SETUP*})
               lcradmin release $endPoint > /dev/null
          fi
     done
}

WaitForDaemons() {
     while true
     do
          for pid in "$@"
          do
               shift
               if kill -0 "$pid" 2>/dev/null
               then
                    set -- "$@" "$pid"
               elif ! wait "$pid"
               then
                    error="true"
               fi
          done
          (("$#" > 0)) || break
          sleep ${WAITALL_DELAY:-1}
     done
}


#Functions for HTML result file
CreateHtmlHead()
{
cat > "$resultHtml" <<EOF
<html>
  <head>
    <!--Load the AJAX API-->
    <script type="text/javascript"
src="https://www.google.com/jsapi"></script>
    <script type="text/javascript">
      google.load('visualization', '1.0',
{'packages':['corechart']});
      google.setOnLoadCallback(drawChart);
      function drawChart() {
      }
    </script>
  </head>
  <title>Telephony Test Results</title>
  <body>
```

```
    <h2>Test started at: $fileDate on device: $device
$firmVersion</h2>
    <table>
waiting for next
EOF
}

ConfigChart()
{
sed -i "s_function drawChart.*_\
function drawChart\(\) { \n\
        var data$group = new
google.visualization.DataTable\(\)\n\
        data$group.addColumn\(\'string\', \'Title\'\)\n\
        data$group.addColumn\(\'number\', \'Value\'\)\n\
        data$group.addRows\(\[\n\
          \[\'Successful Calls\', $successfulCallsGroup\],\n\
          \[\'Failed Calls\', $failedCallsGroup\]\n\
        \]\);\n\
        var options$group = {\'title\':'Results\'};\n\
        var chart$group = new
google.visualization.PieChart(document.getElementById\(\'piec
hart$group\'\)\);\n\
        chart$group.draw\(data$group, options$group\);\n\
_g" "$resultHtml"
}

AddTableHead()
{
sed -i "s_waiting for next_\
    <tr>\n\
    <td>\n\
    <h5>Group $group</h5>\n\
    <table border=\"1\" style=\"width:700px; font-size:
10px;\">\n\
    <tr>\n\
      <th>Caller Number</th>\n\
      <th>Callee Number</th>\n\
      <th>Number of Calls</th>\n\
      <th>Length of Calls</th>\n\
      <th>Pause Between Calls</th>\n\
      <th>Incoming/Outgoing</th>\n\
      <th>Average Establish Time</th>\n\
    </tr>\n\
waiting for next\n\
_g" "$resultHtml"
}

AddTable()
{
sed -i "s_waiting for next_\
```

```
    <tr>\n\
      <td>$fromNumber</td>\n\
      <td>$toNumber</td>\n\
      <td>Number of calls $parallelNum $group</td>\n\
      <td>$lengthOfCalls</td>\n\
      <td>$pauseBetweenCalls</td>\n\
      <td>$inOut</td>\n\
      <td>Establish time $parallelNum $group</td>\n\
    </tr>\n\
waiting for next\n\
_g" "$resultHtml"
}

AddGroupResutls()
{
sed -i "s_waiting for next_\
    <tr>\n\
      <td align="center" colspan="7">\
     Successful=$successfulCallsGroup \
     Failed=$failedCallsGroup Average Establish \
     Time=$establishTime</td>\n\
    </tr>\n\
waiting for next\n\
_g" "$resultHtml"
}

AddChart()
{
sed -i "s_waiting for next_\
    </table>\n\
    </td>\n\
    <td>\n\
    <div id=\"piechart$group\" style=\"width: 500px; \
    height: 180px;\"></div>\n\
    </td>\n\
    </tr>\n\
waiting for next\n\
_g" "$resultHtml"
}

AddResults()
{
group=0
successfulCallsGroup=$successfulCalls
failedCallsGroup=$failedCalls
ConfigChart
sed -i "s_waiting for next_\
    </table>\n\
    <div id=\"piechart0\" style=\"width: 800px; height:
400px;\"></div>\n\
  </body>\n\
```

```
</html>\n\
_g" "$resultHtml"
}

#Check if Asterisk and LCR is running
CheckProcess "asterisk"
CheckProcess "lcr"
error="false"

echo "Enter device name (if it has not changed, leave \
the field blank and press ENTER): "
read device
if [ -z "$device" ]
then
     device=$(sed '1!d' $saveFile)
else
     echo $device > $saveFile
fi
echo "Enter firmware version (if it has not changed, leave \
the field blank and press ENTER): "
read firmVersion
if [ -z "$firmVersion" ]
then
     firmVersion=$(sed '2!d' $saveFile)
else
     echo $device > $saveFile
     echo $firmVersion >> $saveFile
fi

echo "Set number of test runs: "
read cycles

for cycle in `seq 1 $cycles`
do

#Rotate asterisk log file
asterisk -rx "logger rotate"

fileDate=`date +"%F_%H-%M-%S"`

#Clear DUT log file
> $dutLogFile

#Dir for results
resultDir="$HOME/TelTest_Results/$device
$firmVersion/Test_$fileDate"
outputFile="$resultDir/Teltest_log_$fileDate.txt"
resultHtml="$resultDir/Result_$fileDate.html"
resultTxt="$resultDir/Result_$fileDate.txt"
partResultFile="$resultDir/part_result_$fileDate"
```

```
#Create outputfile and folder for results
if [ ! -d "$HOME/TelTest_Results/" ]
then
     mkdir "$HOME/TelTest_Results/"
fi
if [ ! -d "$HOME/TelTest_Results/$device $firmVersion/" ]
then
     mkdir "$HOME/TelTest_Results/$device $firmVersion/"
fi
if [ ! -d "$resultDir" ]
then
     mkdir "$resultDir"
fi
echo "Test started at: $fileDate on device: $device
$firmVersion" |tee -a "$outputFile"

> "$resultTxt"

#Start time
start=`date +%s`

CreateHtmlHead

for group in `seq 1 99`
do
     parallelCalls=0
     parallelNum=0
     successfulCallsGroup=0
     failedCallsGroup=0
     establishTime=0
     echo "[from-lcr]" > $asteriskExtensionsFile

     #Read number of parallel calls in current group
     while read line
     do
          echo "$line" |egrep -q "^[[:cntrl:] ]*[#;]|^$|^#" \
          && continue    #Skip comments and empty lines
          set $line
          if [ $7 -eq $group ]
          then
               parallelCalls=$((parallelCalls + 1))
               if [ "$6" = "in" ]
               then
                    echo "exten = $2,1,Answer() " \
                    >> $asteriskExtensionsFile
                    echo "same = n,Wait(99999999)" \
                    >> $asteriskExtensionsFile
                    echo "same = n,Hangup()" \
                    >> $asteriskExtensionsFile
               fi
          fi
```

```
done <$configFile

#Break the process if no more calls found
if [ $parallelCalls -eq 0 ]
then
     continue
fi

echo "Reloading Asterisk dialplan..."
asterisk -rx 'dialplan reload'

#Kill the subprocesses if ctrl+c
trap "Stop" 2

AddTableHead
pids=""

#Create subprocess for every parallel call
while read line
do
     echo "$line" |egrep -q "^[[:cntrl:] ]*[#;]|^$|^#" \
     && continue    #Skip comments and empty lines
     set $line
     if [ $7 -eq $group ]
     then
          parallelNum=$((parallelNum + 1))
          fromNumber=$1
          toNumber=$2
          numberOfCalls=$3
          lengthOfCalls=$4
          pauseBetweenCalls=$5
          inOut=$6
          AddTable
          sh call_daemon.sh "$line" $parallelNum \
          $parallelCalls $group $fileDate "$resultDir" &
          pids="$pids $!"
     fi
done <$configFile

#Wait untill call daemonds are finished
WaitForDaemons $pids

#Get data from subsrcipt
for  i in `seq 1 $parallelCalls`
do
if [ -f "$partResultFile.$i" ]
then
     while read line
     do
          if echo "$line" |grep -q "successfulCalls="
          then
```

```
                    line=$(echo ${line#*successfulCalls=})
                    successfulCallsNew=$line
               fi
               if echo "$line" |grep -q "failedCalls="
               then
                    line=$(echo ${line#*failedCalls=})
                    failedCallsNew=$line
               fi
               if echo "$line" |grep -q "errors="
               then
                    line=$(echo ${line#*errors=})
                    errors=$((errors+line))
               fi
               if echo "$line" |grep -q "establishTime="
               then
                    line=$(echo ${line#*establishTime=})
                    establishTime=$(echo "scale=1; \
                    ($establishTime + $line)" |bc -l)
                    sed -i "s/Establish time $i \
                    $group/$line/g" "$resultHtml"
               fi
          done <"$partResultFile.$i"
          sed -i "s/Number of calls $i $group/\
          $((failedCallsNew + successfulCallsNew))/g" \
          "$resultHtml"
          successfulCallsGroup=$((successfulCallsGroup \
          + successfulCallsNew))
          failedCallsGroup=$((failedCallsGroup + \
          failedCallsNew))
     fi
     done
     establishTime=$(echo "scale=1; \
    ($establishTime / $parallelCalls)" |bc -l)
     successfulCalls=$((successfulCalls + \
     successfulCallsGroup))
     failedCalls=$((failedCalls + failedCallsGroup))
     echo "Group=$group: Success=$successfulCallsGroup \
     Failed=$failedCallsGroup Establish=$establishTime" >> \
     "$resultTxt"
     AddGroupResutls
     ConfigChart
     AddChart
     cp $dutLogFile "$resultDir/dut_log_$fileDate.log.$group"
     > $dutLogFile

     if [ "$error" = "true" ]
     then
          break
     fi
done
```

```
#Copy used part of DUT log file to results
cp teltest.conf "$resultDir/teltest_$fileDate.conf"
cp "/var/log/asterisk/messages"
"$resultDir/asterisk_log_$fileDate.log"

#Calculation of percentage of success and of the elapsed time
percent=$(echo "scale=5; (($successfulCalls / \
($failedCalls + $successfulCalls)) * 100)" |bc -l)
stop=`date +%s`          #Stop time
duration=$(($stop - $start))
hours=$(( $duration / 3600))
minutes=$(( (($duration - ($hours * 3600)) / 60) ))
seconds=$(($duration - ($hours * 3600) - (minutes * 60) ))

echo -e "\nTest started at: $fileDate on device: $device \
$firmVersion\n\
$(($successfulCalls + $failedCalls)) calls were made in: \
$hours:$minutes:$seconds\n\
Successful: $successfulCalls\n\
Failed: $failedCalls\n\
Success: $percent %\n\
Crashes: $errors" |tee -a "$outputFile"

AddResults

#Open results in browser
firefox "$resultHtml" &
if [ "$error" = "true" ]
then
     echo "Test interrupted"
     break
fi
done
```

B Call Daemon

```
#!/bin/sh

line=$1
parallelNum=$2
parallelCalls=$3
group=$4
fileDate=$5
resultDir="$6"

sleepAfterFail=0.1

#Pathes to used files
asteriskLogFile="/var/log/asterisk/messages"
dutLogFile="/var/log/dut_log.log"
lcrLogFile="/usr/local/var/log/lcr/log"
callFile="call_file.call.source"
outputFile="$resultDir/Teltest_log_$fileDate.txt"
asteriskExtensionsFile="/etc/asterisk/extensions.conf"
tempCallFile="temp_file$parallelNum.call"
partResultFile=\
"$resultDir/part_result_$fileDate.$parallelNum"

#Default values
prevLine="empty"
successfulCalls=0
failedCalls=0
errors=0
sleepTime=0.05
establishTimeTotSec=0
establishTimeTotNano=0
releaseTimeTotSec=0
releaseTimeTotNano=0


ExportToLog()
{
     currentTime=`date +"%F_%H:%M:%S"`
     echo "Call: $i/$parallelNum/$group [$currentTime] \
     $inOut from: $fromNumber to: $toNumber length: \
     $lengthOfCalls s pause: $pauseBetweenCalls s \
     Status: $1" | tee -a "$outputFile"
}

ReadConfigFile()
{
     set $line
     fromNumber=$1
```

```
     toNumber=$2
     numberOfCalls=$3
     lengthOfCalls=$4
     pauseBetweenCalls=$5
     inOut=$6
}

MakeCalls()
{
     for  i in `seq 1 $numberOfCalls`
     do
          #Get info for the logfile
          ExportToLog "Making attempt"
          CheckDUT
          callResult="false” #False until it detects
                             #successful or unsuccessful call
          callIdChecked="false"
          echo "Call: $i/$parallelNum/$group [$currentTime] \
          $inOut from: $fromNumber to: $toNumber" >> \
          /var/log/dut_log.log
          cp $callFile $tempCallFile
          SetParameters
          mv $tempCallFile /var/spool/asterisk/outgoing/
          startMeasureTimeSec=`date +%s`
          startMeasureTimeNano=`date +%1N`
          ExportToLog "Making a call"

          if [ "$inOut" = "out" ]
          then
               CheckOutgoingResult
          fi

          if [ "$inOut" = "in" ]
          then
               CheckIncomingResult
          fi

          #Export resutls into temp file
          echo "successfulCalls=$successfulCalls" > \
          "$partResultFile"
          echo "failedCalls=$failedCalls" >> \
          "$partResultFile"
          echo "errors=$errors" >> "$partResultFile"
          echo "establishTime=$establishTimeAvrSec" >> \
          "$partResultFile"

          sleep $pauseBetweenCalls
     done
}
```

```
CheckOutgoingResult()
{
     while true
        do
          lastLines=$(tail -$((parallelCalls * 20)) \
          $asteriskLogFile)
          if [ "$callIdChecked" = "false" ] \
          && echo "$lastLines" \
          |grep -q "\[call=.* ast=lcr.*\] Sending setup to \
          LCR. (interface=ast dialstring=$toNumber, cid=)"
          then
               SetIdNumbersOut
          fi
          #Check for successful call
          if [ "$callResult" = "false" -a \
          "$callIdChecked" = "true" ] \
          && echo "$lastLines" \
          |grep -q "\[call=$callId ast=lcr/$lcrId\] \
          Incomming connect (answer) from LCR"
          then
               MeasureTime
               ExportToLog "Call successfully established"
               startCallTime=`date +%s`
               callResult="success"
          fi

          #Check for failed call
          if [ "$callResult" = "false" -a \
          "$callIdChecked" = "true" ] \
          && echo "$lastLines" \
          |grep -q "\[call=$callId ast=lcr/$lcrId\] \
          Incomming disconnect"
          then
               ExportToLog "Call failed"
               echo "\033[1;31mCall failed\033[0m"
               callResult="fail"
               failedCalls=$((failedCalls + 1))
               ReleaseCall
               sleep $sleepAfterFail
          fi

          #Check for free channel
          if [ "$callIdChecked" = "true" ] \
          && echo "$lastLines" \|grep -q \
          "\[call=$callId ast=NULL\] Call instance freed\|\
          \[call=0 ast=lcr/$lcrId\] Freeing call instance"
          then
               if [ "$callResult" = "success" ]
               then
                    CheckCallLength
               fi
```

```
                ExportToLog "Call instance freed"
                if [ "$callResult" = "fail" ]
                then
                     sleep $lengthOfCalls
                fi
                break
           fi
           sleep $sleepTime
     done
}

CheckIncomingResult()
{
     while true
        do
          lastLines=$(tail -$((parallelCalls * 20)) \
          $asteriskLogFile)
          if [ "$callIdChecked" = "false" ] \
          && echo "$lastLines" \
          |grep -q "\[call=.* ast=lcr.*\] \
          Try to start pbx. (exten=$toNumber"
          then
               SetIdNumbersIn
          fi
          #Check for successful call
          if [ "$callResult" = "false" -a \
          "$callIdChecked" = "true" ] \
          && echo "$lastLines" |grep -q \
          "\[call=$callId ast=lcr/$lcrId\] Starting call"
          then
               MeasureTime
               ExportToLog "Call successfully established"
               startCallTime=`date +%s`
               callResult="success"
          fi

          #Check for failed call
          if [ "$callResult" = "false" ] \
          && echo "$lastLines" |grep -q \
         "\[.*\] NOTICE.* Queued call to SIP/$toNumber.* \
         expired without completion"
          then
               CheckNoticeDate
               if [ $noticeDate -gt $prevNoticeDate ]
               then
                    noticeDate=$prevNoticeDate
                    ExportToLog "Call failed"
                    echo "\033[1;31mCall failed\033[0m"
                    callResult="fail"
                    failedCalls=$((failedCalls + 1))
                    sleep $lengthOfCalls
```

```
                    sleep $sleepAfterFail
                    break
               fi
          fi

          #Check for free channel
          if [ "$callIdChecked" = "true" ] \
          && echo "$lastLines" |grep -q \
          "\[call=0 ast=lcr/$lcrId\] Freeing call instance"
          then
               if [ "$callResult" = "success" ]
               then
                    CheckCallLength
               fi
               ExportToLog "Call instance freed"
               CheckNoticeDate
               prevNoticeDate=$noticeDate
               break
          fi

          sleep $sleepTime
     done
}

CheckLastId()
{
     for j in `seq 1 $((parallelCalls * 20))`
     do
          lastLines=$(tail -$j $asteriskLogFile)
          if [ "$inOut" = "out" ]
          then
               if echo "$lastLines" |grep -q \
               "\[call=.* ast=lcr.*\] Sending setup to LCR. \
               (interface=ast dialstring=$toNumber, cid=)"
               then
                    prevId=$(echo "$lastLines" |grep \
                    "\[call=.* ast=lcr.*\] Sending setup to \
                    LCR. (interface=ast dialstring=\
                    $toNumber, cid=)")
                    prevId=$(echo ${prevId##*call=})
                    prevId=$(echo ${prevId% ast*})
                    break
               fi
          fi
          if [ "$inOut" = "in" ]
          then
               if echo "$lastLines" \
               |grep -q "\[call=.* ast=lcr.*\] \
               Try to start pbx. (exten=$toNumber"
               then
                    prevId=$(echo "$lastLines" \
```

```
                    |grep "\[call=.* ast=lcr.*\] \
                    Try to start pbx. (exten=$toNumber")
                    prevId=$(echo ${prevId##*call=})
                    prevId=$(echo ${prevId% ast*})
                    break
               fi
               if echo "$lastLines" \
               |grep -q "\[.*\] NOTICE.* Queued call to \
               SIP/$toNumber.* expired without completion"
               then
                    prevNoticeDate=$(echo "$lastLines"|grep \
                    "\[.*\] NOTICE.* Queued call to SIP\
                    /$toNumber.*expired without completion")
                    month=`date +%b`
                    prevNoticeDate=$(echo \
                    ${prevNoticeDate##*\[$month })
                    prevNoticeDate=$(echo \
                    ${prevNoticeDate%\] NOTICE*})
                    break
               fi
          fi
     done
}

SetIdNumbersOut()
{
     callId=$(echo "$lastLines" \
     |grep ".*\[call=.* ast=lcr.*\] Sending setup to LCR. \
     (interface=ast dialstring=$toNumber, cid=)")
     lcrId=$callId
     callId=$(echo ${callId##*call=})
     callId=$(echo ${callId% ast*})
     if [ "$prevId" != "$callId" ]
     then
          prevId=$callId
          lcrId=$(echo ${lcrId##*ast=lcr\/})
          lcrId=$(echo ${lcrId%\] Sending*})
          callIdChecked="true"
     fi
}

SetIdNumbersIn()
{
     callId=$(echo "$lastLines" \
     |grep "\[call=.* ast=lcr.*\] Try to start pbx. \
     (exten=$toNumber")
     lcrId=$callId
     callId=$(echo ${callId##*call=})
     callId=$(echo ${callId% ast*})
     if [ "$prevId" != "$callId" ]
     then
```

```
          prevId=$callId
          lcrId=$(echo ${lcrId##*ast=lcr\/})
          lcrId=$(echo ${lcrId%\] Try to*})
          callIdChecked="true"
     fi
}

#Setting new parameters for calls
SetParameters()
{
     sed -i "s/Data: .*/Data: $lengthOfCalls/g" \
     $tempCallFile
     sed -i "s/CallerID: .*/CallerID: \"Asterisk\" \
     <$fromNumber>/g" $tempCallFile
     if [ "$inOut" = "out" ]
     then
          sed -i "s/Channel: LCR\/ast\/.*/Channel: \
          LCR\/ast\/$toNumber/g" $tempCallFile
     fi
     if [ "$inOut" = "in" ]
     then
          sed -i "s/Channel: LCR\/ast\/.*/Channel: \
          SIP\/$toNumber@192.168.115.24/g" $tempCallFile
     fi
}

CheckCallLength()
{
     callTime=`date +%s`
     callTime=$((callTime - startCallTime))
     lengthOfCallsInt=$(echo "$lengthOfCalls/1" |bc)
     callDifference=$((lengthOfCallsInt - callTime))
     if [ $callDifference -lt 0 ]
     then
          callDifference=$((0 - callDifference))
     fi

     if [ $callDifference -gt 10 -a $callDifference -gt \
     $((lengthOfCallsInt / 1000)) ]
     then

          ExportToLog "Wrong call length"
          failedCalls=$((failedCalls + 1))
          echo "\033[1;31mCall failed\033[0m"
     else
          ExportToLog "Correct call length"
          echo "\033[1;92mCall successful\033[0m"
          successfulCalls=$((successfulCalls + 1))
     fi
}
```

```
MeasureTime()
{
     measureTimeSec=`date +%s`
     measureTimeNano=`date +%1N`
     measureTimeSec=$((measureTimeSec - startMeasureTimeSec))
     measureTimeNano=$((measureTimeNano - \
     startMeasureTimeNano))
     establishTimeTotSec=$((measureTimeSec + \
     establishTimeTotSec))
     establishTimeTotNano=$((measureTimeNano + \
     establishTimeTotNano))
     establishTimeAvrNano=$(echo "scale=1; \
     ($establishTimeTotNano / ($i * 10))" |bc -l)
     establishTimeAvrSec=$(echo "scale=1; \
     (($establishTimeTotSec / $i) + $establishTimeAvrNano)" \
     |bc -l)
}

ReleaseCall()
{
     for k in `seq 1 $((parallelCalls * 20 * lengthOfCalls))`
     do
          lcrLines=$(tail -$k $lcrLogFile)
          if echo "$lcrLines" \
          |grep -q "EP(.*): SETUP  from CH(.*) \
          interface from=ast .*  dialing $toNumber"
          then
               endPoint=$(echo "$lcrLines" \
               |grep "EP(.*): SETUP  from CH(.*) \
               interface from=ast .*  dialing $toNumber")
               endPoint=$(echo ${endPoint#*EP\(})
               endPoint=$(echo ${endPoint%\)\: SETUP*})
               lcradmin release $endPoint > /dev/null
               break
          fi

     done
}

CheckNoticeDate()
{
     if [ "$callResult" = "success" ]
     then
          noticeDate=$(echo "$lastLines" |grep \
          "\[call=0 ast=lcr/$lcrId\] Freeing call instance")
     else
          noticeDate=$(echo "$lastLines" |grep \
          "\[.*\] NOTICE.* Queued call to \
          SIP/$toNumber.*expired without completion")
     fi
     month=`date +%b`
```

```
     noticeDate=$(echo ${noticeDate##*\[$month })
     noticeDate=$(echo ${noticeDate%\] NOTICE*})
     noticeDate=$(echo "$noticeDate" |tr -d " ")
     noticeDate=$(echo "$noticeDate" |tr -d ":")
     echo "$noticeDate > $prevNoticeDate"
}

WaitForReboot()
{
     while true
     do
          dutLog=$(tail -200 $dutLogFile)
          echo "$dutLog" |grep -q "User is successfully \
          registred" && break
          sleep 1
     done
     sleep 1
}

CheckDUT()
{
     #Check state of DUT
     dutLog=$(tail -500 $dutLogFile)
     if echo "$dutLog" |grep -q "Fatal exception: panic"
     then
          currentTime=`date +"%F_%H:%M:%S"`
          echo "Error [$currentTime] DUT crash" \
          |tee -a "$outputFile"
          echo "DUT reboots, waiting..."
          errors=$((errors + 1))
          sleep 120
          WaitForReboot
     fi
}

ReadConfigFile

#Check call ID from previous calls
CheckLastId

MakeCalls
exit 0
```